In this chapter you will learn to:

- identify an appropriate language to solve a particular problem
- recognise the appropriateness of either a sequential or event-driven approach to solve a particular problem
- develop syntactically correct code to solve a problem in a given language
- interpret metalanguage definitions for commands in a selected language
- produce syntactically correct statements using the metalanguage definitions
- produce a generic metalanguage definition for a set of syntactically correct statements that use the same command
- implement a solution from a complex algorithm using syntactically correct statements
- explain the use of tokens and the role of the parsing process during the translation of source code to machine code
- recognise that machine code is the only code able to be executed by a computer
- identify the most appropriate translation method for a given situation
- use the features of both a compiler and an interpreter in the implementation of a software solution
- recognise, interpret and write machine code instructions for a problem fragment
- employ good programming practice when developing code
- justify the use of a clear modular structure with separate routines to ease the design and debugging process
- differentiate between types of errors, recognise the cause of a specific error and determine how to correct it
- effectively use a variety of appropriate error correction techniques to locate the cause of a logic error and then correct it
- produce user documentation (incorporating screen dumps) that includes a user manual, a tutorial, online help
- differentiate between types of user documentation
- identify the personnel who would be likely to use the different types of documentation
- produce technical documentation for an implemented software solution
- recognise the need for additional hardware
- identify potential compatibility issues for a newly developed software solution
- recognise the implications of emerging technologies for the developer in terms of the code written to make use of these technologies
- recognise the implications of emerging technologies for the code development process

Which will make you more able to:

- explain the interrelationship between hardware and software
- differentiate between various methods used to construct software solutions
- describe how the major components of a computer system store and manipulate data
- explain the implications of the development of different languages
- explain the relationship between emerging technologies and software development
- identify and evaluate legal, social and ethical issues in a number of contexts
- construct software solutions that address legal, social and ethical issues
- apply appropriate development methods to solve software problems
- apply a modular approach to implement well structured software solutions and evaluates their effectiveness
- apply project management techniques to maximise the productivity of the software development
- create and justify the need for the various types of documentation required for a software solution
- selects and applies appropriate software to facilitate the design and development of software solutions
- communicate the processes involved in a software solution to an inexperienced user
- use and describe a collaborative approach during the software development cycle
- develop and evaluate effective user interfaces, in consultation with appropriate people

In this chapter you will learn about

Implementation of the design using an appropriate language

- the different programming languages and the appropriateness of their use in solving different types of problems
- construction of syntactically correct code that implements the logic described in the algorithm

Language syntax required for software solutions

- use of EBNF and railroad diagrams to describe the syntax of statements in the selected language

The need for translational to machine code from source code

- translation methods in software solutions including compilation and interpretation
- advantages and disadvantages of each method
- steps in the translation process
  - lexical analysis including token generation
  - syntactical analysis including parsing
  - code generation

The role of machine code in the execution of a program

- machine code and CPU operation
  - instruction format
  - use of registers and accumulators
  - the fetch–execute cycle
  - use of a program counter and instruction register
- execution of called routines
- linking, including use of DLLs

Techniques used in developing well-written code

- the use of good programming practice, including:
  - a clear and uncluttered mainline
  - one logical task per subroutine
  - use of stubs
  - appropriate use of control structures and data structures
  - writing for subsequent maintenance
  - version control and regular backup
  - recognition of relevant social and ethical issues
- the process of detecting and correcting errors, including:
  - types of error
    - syntax errors and logic errors
    - runtime errors, including arithmetic overflow, division by zero, accessing inappropriate memory locations
  - methods of error detection and correction
    - use of flags
    - methodical approach to the isolation of logic errors
    - use of debugging output statements
    - peer checking, desk checking, structured walkthrough
    - comparison of actual with expected output
- the use of software debugging tools, including use of breakpoints, resetting variable contents, program traces, single line stepping

Documentation of a software solution

- forms of documentation, including:
  - log book
  - user documentation, including user manual, reference manual, installation guide, tutorial, online help
  - technical documentation, including systems documentation, algorithms, source code
- use of application software including CASE tools to assist in the documentation process
- recognition of relevant social and ethical issues

Hardware environment to enable implementation of the software solution

- hardware requirements including minimum configuration, possible additional hardware, appropriate device drivers or extensions

Emerging technologies

- the effect of emerging hardware and software technologies on the development process, such as iPhone, Wii remote, handheld communication devices, scanning pen, biometric devices, multi-point surface software, radio frequency identification (RFID), social networking software.

5

IMPLEMENTATION OF SOFTWARE SOLUTIONS

Implementation of software solutions is the third stage of the software development cycle. During this stage of development the source code is written and tested. That is, the plans and designs formulated in stage two are implemented in a programming language.

We first briefly examine a variety of different programming languages with an emphasis on their appropriateness for solving different types of problems. The syntax of high-level programming languages is then examined using metalanguages such as EBNF and railroad diagrams. This information will allow for the creation of syntactically correct source code.

The source code is translated into machine code. We examine this process of translation including compilation and interpretation. The machine code or object code is then executed by the CPU. This process of execution is studied, with particular reference to the components and actions occurring within the CPU. These actions result in the execution of machine code commands.

Program development techniques are examined. This section describes how well structured, modularised and documented source code should be created. It also explains methods of error detection and correction.

Documentation of the final solution for various audiences is vital for the ongoing support of a software product. We examine a number of different forms of documentation for users and future developers.

Software solutions must operate with hardware and existing software. We consider the hardware and software requirements of new software products. Finally, we examine some emerging technologies and trends likely to affect future software development.

CHOOSING AN APPROPRIATE PROGRAMMING LANGUAGE

There are numerous factors to consider when selecting an appropriate programming language for a specific project. For some projects there will be little choice due to the nature of the targeted environment and device. When Apple's iPhone and iPad were released programmers were limited to Objective C, C++ or C programming languages and similarly Android developers where limited to Java. Although (in 2012) these remain the languages of choice for the development of most smart phone apps many other languages can be used. For software applications running on desktops the range of languages is extensive, however rightly or wrongly it is often the experience of the developers that becomes the overriding selection criteria. Web-based software commonly contains two separate elements – the server side code and the client side code which often runs within the user's browser. In most cases one language is used for the server side and quite a different language is used for the client side code. For instance, PHP is commonly used on the server and JavaScript on the client side.

When choosing a programming language a variety of different criteria should be considered including:

- The nature of the project. Different languages are suited to the solution of different types of problems. An artificial intelligence application will be suited to a different language compared to software to run an online store. The best language to write a low level hardware driver will be very different to the best language to write a platform game. Research is required to determine the most suitable language for the type of software project.
- The intended environment including hardware, operating system and network use has a large impact on the most suitable language. There are interpreted languages designed for use on web servers and others which operate within client web browsers. Some languages compile down to an intermediate code (such as Java's byte code) and will execute on multiple different operating systems. Other languages or language variants are designed for smart phones or tablets.
- The experience of the programmers. It takes a significant amount of time and effort to learn a new language and to become an expert requires experience which can only be obtained over time. Therefore, it is often better to select a language the developer's are already familiar with rather than choose a new language despite the new language being more appropriate for a particular project.
- Event driven or sequential approach. Software applications which require a graphical user interface (GUI) often use an event driven approach. This means subroutines within the software execute in response to events initiated by the user, such as clicking a button or selecting from a menu. Other events such as an incoming network message or a sensor detecting movement are also events the system can detect. The design of event driven code is different to sequential code as there is no definite start and end. A sequential program has a single start, the processing commences and then finally the program ends. The code controls the order of execution, which includes directing the user through the logic. Event driven programs are controlled and led by the user. As the user initiates commands the program runs the code associated with that event.
- Maintainability. Does the language encourage or enforce good practice in terms of well structured and self contained modules which are easy to maintain and reuse? Is there a large community of programmers and a large library of code which can be examined for ongoing support?

Consider the following:

Over a number of years the following languages have been consistently ranked within the top 20 according to a variety of different surveys:

- Java
- JavaScript
- C
- C#
- C++
- Objective-C
- PHP
- Visual Basic (including VB.NET)
- Python
- Delphi
- Pascal
- Lisp
- Ruby
- Perl

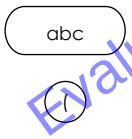
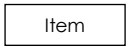
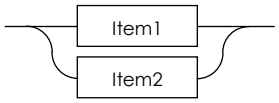
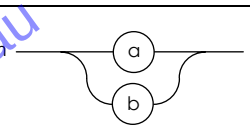
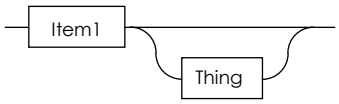
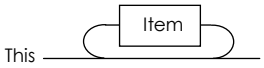
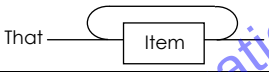
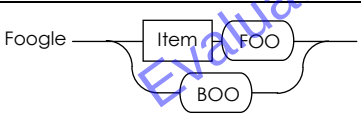
GROUP TASK Research
Each class member is to research the general nature of a few of the above languages. Share a summary for each language with the class.

LANGUAGE SYNTAX REQUIRED FOR SOFTWARE SOLUTIONS

In this section, we examine the syntax of programming languages. The syntax of programming languages is described using metalanguages. In the Preliminary course, we examined EBNF and railroad diagrams. We will use these metalanguages to describe the syntax of commands used to define and use multi-dimensional arrays, arrays of records, files and random number generators.

USE OF EBNF AND RAILROAD DIAGRAMS

Before we begin using these metalanguages to describe the syntax of programming languages, let us revise the syntax of the metalanguages themselves:

Interpretation	EBNF example	Railroad Diagram example
Terminal Symbol for a reserved word. BEGIN is a reserved word.	BEGIN	
Terminal symbol for a literal. The characters abc are written as they appear. Quotes are used to enclose symbols used by the metalanguage.	abc "("	
Non-terminal symbol. Item is defined elsewhere.	<Item>	
"or" a choice between two alternatives. Either Item1 or Item2.	<Item1> | <Item2>	
"is defined as".Item can take the value a or b	Item=a | b	
Optional part. Item followed optionally by a Thing.	<Item>[<Thing>]	
Possible repetition. This is an Item repeated zero or more times.	This={<Item>}	
Repetition. That is an Item repeated one or more times.	That=<Item>{<Item>}	
Grouping. A Foogle is an Item followed by the reserved word FOO or it is the reserved word BOO.	Foogle=(<Item>FOO) | BOO	

GROUP TASK Activity
Create an EBNF and railroad diagram for a Boogle. A Boogle always begins with an Item, followed by one of more Foogles and can either end with a That or a This. Create a series of five legitimate Boogles using your metalanguage descriptions.

METALANGUAGE DESCRIPTIONS OF PROGRAMMING LANGUAGES

Let us examine metalanguage descriptions for defining and using multi-dimensional arrays, arrays of records, files and random number generators in a variety of high-level languages.

Declaring multi-dimensional arrays and arrays of records

In general, declaring a variable for use within a program involves two stages. Firstly, if the data type or structure does not already exist in the language then it must be defined. Secondly, a variable of the required type is declared.

EBNF diagrams to perform these actions are described below:

Visual Basic .NET

The Structure statement is used to create a user defined data type.

```
AccessModifier = Dim | Public | Private

Identifier = <Letter> { <Letter> | <Digit> | _ }

DimStatement = <AccessModifier> <Identifier> [ "("<PositiveInteger>")" ] As <Type>

StructureStatement = <AccessModifier> Structure <Identifier>
                         <DimStatement>
                         { <DimStatement> }
                     End Structure
```

The Type non-terminal element above can be any of the available types included within Visual Basic (String, Integer, Double, Boolean, etc.) or it can be a user defined type that has been declared using a Structure statement.

The description of the DimStatement above indicates that arrays can be declared directly without using the StructureStatement. To declare an array the identifier is followed by the upper bound in brackets, such as Private MyArray(4) As String to declare a string array called MyArray with 5 elements (subscript 0 to 4). However, record data structures require definition using a StructureStatement and then declaration using a DimStatement.

Pascal

The TYPE statement is used to create a user defined data type.

```
DefineDataType=TYPE <varname>=(ARRAY"["<range>{,<range>}"]"OF<type>) |
                    (FILE OF <type>) |
                    (RECORD<varname>:<type>{;<varname>:<type>}END);
```

The VAR statement is used to declare variables of the required types.

```
DeclareDataType=VAR <varname>{,<varname>}:<type>;
                    {<varname>{,<varname>}:<type>;}

varname=<letter>{<letter> | <digit>}
range=<constant>..<constant>
```

GROUP TASK Discussion

Examine the above EBNF diagrams. Discuss the essential differences between the two languages, particularly in regard to declaring arrays.

Note: The EBNF and railroad diagrams used in this chapter explain features necessary for this course. The actual EBNF and railroad productions for these statements include many more features which have been omitted for purposes of clarity.

Using multi-dimensional arrays

Consider the following:

A program requires a multi-dimensional array called Products to be available to all procedures and functions.

The Products array is to have three indexes. The first index ranging from 0 to 10 is used to represent the Supplier. There is a maximum of 11 suppliers. The second index represents one of the five warehouses where the product is stored. The third index determines individual products. It is unlikely there will be more than 100 products from a particular supplier stored in a particular warehouse. Each data item stored in the array will be a string with a maximum of 20 characters.

For example, Products(2, 3, 41)="Widget" means that Widgets are from supplier number 2 and are stored in warehouse 3. Widgets are the 40th product from supplier 2 to be stored in warehouse 3. Note that each index begins at zero hence there is a supplier number 0, a warehouse number 0 and a product 0.

Let us create the products data structure in both Visual Basic and Pascal using the EBNF diagrams from the previous page.

Visual basic

A possible definition for our Products variable could be:

```
Public Products(10, 4, 99) AS String
```

This statement is best placed in the declaration section of a module. The Public keyword ensures that Products is a global variable available to all procedures and functions. The first index has a range of 0 to 10, in Visual Basic 0 is the default for the lower bound so need not be included.

Pascal

A possible definition for our Products variable could be:

```
TYPE
    ProductType=ARRAY[0..19] OF Char;
    ProductArrayType=ARRAY[0..10,0..4,0..99] OF ProductType;
VAR
    Products      : ProductArrayType;
```

In Pascal, strings are not standard data types and must be defined as arrays of characters. Once the array of characters has been declared, it can be used to define other data structures.

GROUP TASK Investigation

Examine another programming language and develop declaration statements for the above Products array.

One of the modules that uses the Products array is described in the IPO Chart below:

IPO CHART

Client: Choog Soft Pty. Ltd.
Project: Warehousing System
Module: NextProductNumber

Date: 29/2/2001
Programmer: Melissa Davis
Page: 15 of 28

Input	Process	Output
Supplier Number, Warehouse Number	Examine each product in turn with Supplier Number and Warehouse number.	
	If Product is empty then this Index provides the next available Product Number.	
	Return the product number to the calling module. Return –1 if no space available.	NextProductNumber

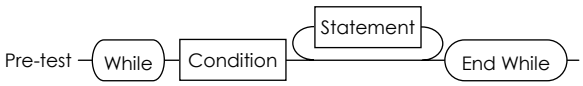
Fig 5.1
IPO Chart for the FindProductNumber function.

Melissa has already developed an algorithm for this function. Her algorithm is shown in *Fig 5.2*.

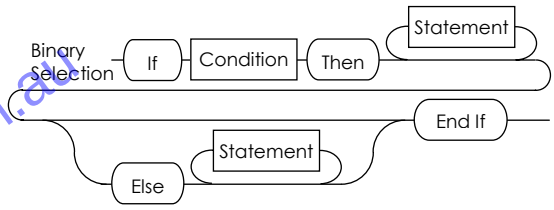
Let's implement the NextProductNumber function in both Visual Basic and Pascal.

Visual Basic .NET

A railroad diagram for the pre-test loop structure in Visual basic is:



For binary selection the Visual Basic railroad diagram is:



In Visual Basic the function could be written as follows:

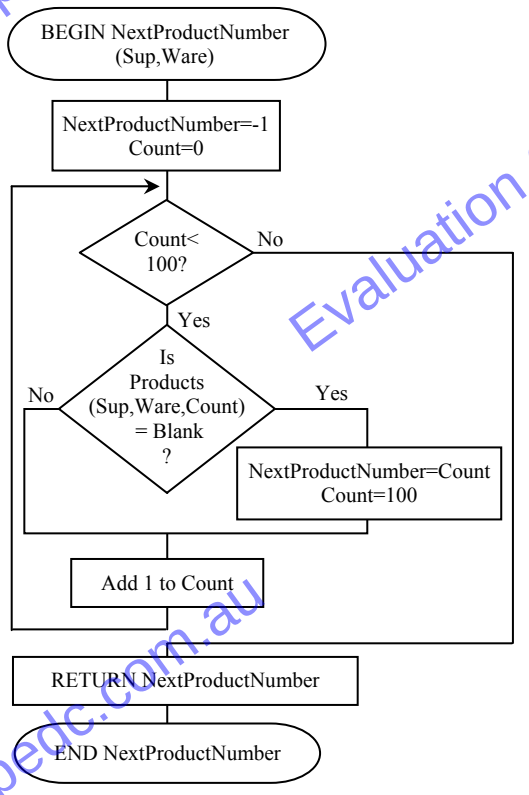


Fig 5.2
Algorithm for the FindProductNumber function.

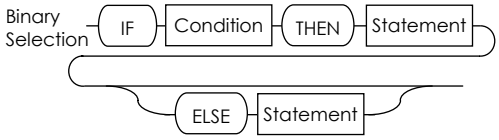
```
Function NextProductNumber(ByVal sup As Integer, ByVal ware As Integer) As Integer
   Dim count As Integer
   NextProductNumber = -1
   count = 0
   While count < 100
      If Products(sup, ware, count) = "" Then
         NextProductNumber = count
         count = 100
      End If
      count = count + 1
   End While
   Return NextProductNumber
End Function
```

GROUP TASK
Activity

Create some sample data for the products array. Complete a desk check of the Visual Basic code. Explain, in words, the processing taking place. This function is similar to one of the searches from chapter 4; explain the similarities.

Pascal

Railroad diagrams for the pre-test loop and binary selection structures in Pascal are:



For those readers not familiar with Pascal, a statement can be made up of multiple statements. When multiple statements are used then the keywords BEGIN and END are used to group the statements. Semi-colons are used to separate each statement within a multiple statement.

In Pascal the function could be written as follows:

```
FUNCTION NextProductNumber (Sup : Integer;
                            Ware : Integer) : Integer;
VAR
   Count : Integer;
BEGIN
  FindProductNumber:=-1;
  Count:=1;
  WHILE Count<=100 DO
     BEGIN
       IF Products[Sup,Ware,Count]="" THEN
          BEGIN
            FindProductNumber:=Count;
            Count:=100
          END;
       Count=Count+1
     END
END;
```

GROUP TASK Discussion

The compiler ignores the spacing and indenting in the above Pascal code. The code would operate exactly the same if no spacing or indenting was used. Discuss the reasons for including spacing and indenting in source code.

GROUP TASK Investigation

Examine another programming language and develop the NextProductNumber function in this language.

Using arrays of records and files

Consider the following:

Imagine that you work for Bizzy Soft Pty. Ltd. a software development company that writes business applications. You are currently working on a product to manage customer details for businesses.

The current project requires an array of records called Customers. This array of records needs to store up to 500 customers. The details for each customer include their Surname, First Name, Sex and Email address. Surnames and First Names have a maximum length of 20 characters, Email addresses up to 30 characters and the Sex can hold one of two values.

You have been allocated the task of writing two subroutines for this product. One to save all the customer records to disk and another to retrieve them from disk back into the Customers data structure.

Let us create the Customers data structure in both Visual Basic .NET and Pascal.

Visual Basic .NET

```
Public Structure CustomerType
   Public Surname As String
   Public FirstName As String
   Public Sex As Boolean
   Public Email As String
End Structure

Public Customers(500) As CustomerType
```

However the above CustomerType definition creates strings of variable length. This will work correctly if we use a sequential file, however for a relative (or random) access file we need to create fixed length strings so that each record will always be precisely the same length. The following definition creates fixed length records:

```
Public Structure CustomerType
   <VBFixedString(20)> Public Surname As String
   <VBFixedString(20)> Public FirstName As String
   Public Sex As Boolean
   <VBFixedString(30)> Public Email As String
End Structure

Public Customers(500) As CustomerType
```

Pascal

```
TYPE
   CustomerRecType=RECORD
     Surname    : ARRAY[1..20] OF Char;
     FirstName  : ARRAY[1..20] OF Char;
     Sex        : Boolean;
     Email      : ARRAY[1..30] OF Char
   END;

   CustomerType=ARRAY[1..500] OF CustomerRecType;

VAR
   Customers    : CustomerType;
```

GROUP TASK Investigation
Examine another programming language and develop declaration statements for the above Customers data structure.

We now need to develop the sub-programs to store and retrieve the Customer's records from a disk file. As the developer, you have already created an algorithm to complete the storage task (see *Fig 5.3*). Retrieving the records will use the identical algorithm except instead of writing the records to the file we will be reading them from the file.

```
BEGIN StoreCustomers (FileName)
   Open Filename for output
   Count=0
   WHILE Count<=500
      Write Filename from Customers(Count)
      Add 1 to Count
   ENDWHILE
   Close Filename
END StoreCustomers
```

Fig 5.3
Algorithm to store customer details to file.

Let us now examine the syntax of statements required to complete these file access tasks in Visual Basic.

Visual Basic .NET

The required EBNF diagrams are:

The FileOpen statement is used to open (or create) a file and prepare it for input or output:

OpenFile = FileOpen "(" <FileNumber>, <FileName>, <Mode>, [<Access>], [<Share>], [<RecordLength>] ")"

Mode = Append | Input | Output | Random

Access = Read | Write | Read/Write

Share = LockRead | LockWrite | LockRead/Write | Shared

RecordLength = <integer>

FileName is used to specify the full path including the filename to the file. Mode can be Input, Append or Output for sequential files or Random for random access files. Filenumber is an integer between 1 and 511. Each file opened must have a unique file number. The next free file number can be obtained using the FreeFile function. RecordLength is required for random access files and specifies the length of a record in bytes. The length of a record can be determined using the Len function.

Closing a file is performed with the FileClose statement:

CloseFile= FileClose "(" [<FileNumber> {, <FileNumber>}] ")"

If the FileClose statement is used without a FileNumber then all files which were opened with a FileOpen statement will be closed.

For random access files writing and reading data to and from a file is accomplished with the FilePut and FileGet statements:

WriteToFile = FilePut "(" <FileNumber>, <RecordVariable>, [<RecNumber>] ")"

ReadFromFile = FileGet "(" <FileNumber>, <RecordVariable>, [<RecNumber>] ")"

If RecNumber is omitted then the next record is used each time a FilePut or Fileget is executed. The first record in a file is RecNumber 1.

For sequential files, writing and reading data to and from a file is accomplished with the Write and Input statements. In Visual Basic each field of a record must be written to and read from sequential files explicitly. This was not the case for random files where a complete record is read/written as a complete unit.

WriteToFile = Write "(" <FileNumber>, <Variable> {,<Variable>} ")"

ReadFromFile =Input "(" <FileNumber>, <Variable> {,<Variable>} ")"

Multiple data items can be written to a sequential file with one Write statement. Similarly, multiple data items can be read from a sequential file with one Input statement. Data is stored in the file with commas between each data item and any strings are enclosed in double quotes.

Let us now implement our StoreCustomers and RetrieveCustomers procedures in Visual Basic .NET. We will use a random access file for this purpose, although a sequential file would have fulfilled our requirements equally well.

```
Public Sub StoreCustomers (FileName As String)
Dim Count As Integer, FileNum As Integer, CusLen As Integer
FileNum = FreeFile()
CusLen = Len(Customers(0))
FileOpen ( FileNum, FileName, OpenMode.Random, , , CusLen )
Count = 0
While Count <= 500
   FilePut ( FileNum, Customers(Count), Count+1 )
   Count = Count + 1
End While
FileClose ( FileNum )
End Sub
```

We use the FreeFile() function to obtain the next available file number. The Len function is used to find the length of a customer record. As all the customer records are the same length we can find the length of any record, in this case we've found the length of the first record. These values are used as parameters in the statements that follow.

The RetrieveCustomers procedure could be implemented as follows:

```
Public Sub RetrieveCustomers (FileName As String)
Dim Count As Integer, FileNum As Integer, CusLen As Integer
FileNum = FreeFile
CusLen = Len(Customers(0))
FileOpen ( FileNum, FileName, OpenMode.Random, , , CusLen )
Count = 0
While NOT EOF( FileNum )
   FileGet ( FileNum, Customers(Count), Count+1 )
   Count=Count+1
End While
FileClose ( FileNum )
End Sub
```

The RetrieveCustomers procedure above is almost identical to the StoreCustomers procedure in terms of logic. The condition to end the loop uses the EOF function; this function detects the end of the file specified by FileNum.

Note: The VB.NET sequential and random access commands described above are similar to those described in the Software Design and Development Stage 6 - Software and Course Specifications document. There are other more efficient file access methods provided within the .NET framework.

GROUP TASK Activity

A problem occurs with both procedures above if the file does not exist. In this case the OPEN statement will create a new empty file. The procedures still terminate correctly but an empty file will remain on the disk. The Visual Basic Kill command is used to remove files. The syntax is Kill "(" <Pathname> ")". Alter the procedures to correct this fault.

Consider the following:

Imagine your Year 12 class are running a raffle to raise funds for your farewell. There are 500 tickets to be sold to students at the school. First prize is an iPod, second prize is a music voucher and there are 20 other prizes of a free Mars bar from the canteen.

As a keen Software Design and Development class, you decide to create a VB.NET program to manage the raffle. Different routines have been allocated to different members of your SDD class. Fred (not his real name) has been given the task of writing the routine that creates the winners. Fred is not known for his honesty but he is certainly a capable programmer.

```
Public Structure TicketType
   Public Surname AS String
   Public FirstName AS String
   Public Year AS Integer
End Structure
Public Tickets(500) AS TicketType
```

Fig 5.4
The Tickets data structure for the Year 12 raffle.

The main data structure is an array of records called Tickets. The details of the Tickets data structure are given in *Fig 5.4*. Fred has developed an algorithm for this task and has just completed coding the algorithm in Visual Basic .NET.

```
Public PrizeWinners(22) As Integer

Public Sub Winners()
   Dim Winner As Integer, WinNum As Integer, CheckCounter As Integer
   Dim rand As New Random
   WinNum = 1
   While WinNum <= 22
      Winner = rand.Next(0, 500)
      PrizeWinners(WinNum) = Winner
      CheckWinner(WinNum, Winner)
      CheckCounter = 1
      While CheckCounter < WinNum
         If PrizeWinners(CheckCounter) = Winner Then
            WinNum = WinNum - 1
            CheckCounter = WinNum
         End If
         CheckCounter = CheckCounter + 1
      End While
      WinNum = WinNum + 1
   End While
End Sub
```

Fig 5.5
Fred's Winners VB.NET procedure

GROUP TASK Investigation

Examine the Winners procedure ignoring the call to the CheckWinner procedure. Explain in words how Fred has completed the required task.

```
Public Sub CheckWinner(ByRef A As Integer, ByRef B As Integer)
    Dim rand As New Random
    If Tickets(B).Year < 11 And rand.Next(0, 5) <> 0 Then
        A = A - 1
    ElseIf A < 3 And A + Tickets(B).Year <> 13 Then
        A = A - 1
    End If
End Sub
```

Fig 5.6
Fred's CheckWinner VB.NET procedure

GROUP TASK Investigation
Now include the call to the CheckWinner procedure in your investigation. What has Fred done to alter the outcome of the raffle? What is the significance of Fred's use of ByRef rather than ByVal for the parameters to the procedure?

HSC style questions:

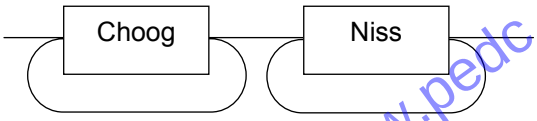
Consider the following EBNF statements when answering Questions 1 and 2.

Choog = A|B|C|D
Niss = a|b|c|d
Ving = {<Choog>} | {<Niss>}
Ruke = <Choog> <Niss> <Ving>

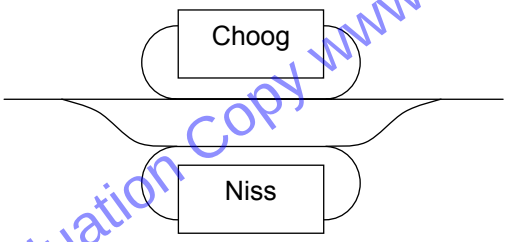
1. Which of the following is a legitimate Ruke?
(A) ABab
(B) AaAaAa
(C) bBccc
(D) CcABCD

2. Which of the following railroad diagrams correctly defines a Ving?

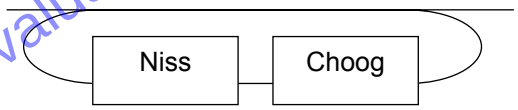
(A)



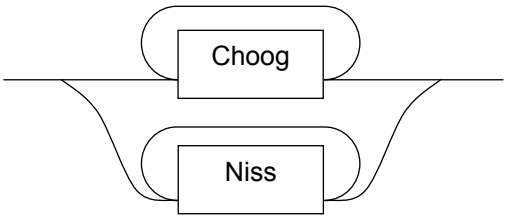
(B)



(C)



(D)



3. In EBNF a Drim is defined as follows:

Drim = (<Yot> | (<Hok> { <Hok> })) <Yot>

Which of the following best describes a Drim?

(A) A Drim begins with a Yot followed by one or more Hoks with a Yot at the end.

(B) A Drim begins with a Yot or a Hok, which is followed optionally by another Hok with a Yot at the end.

(C) A Drim is two Yots or a Drim is a series of one or more Hoks followed by a Yot.

(D) A Drim is a series of Hoks surrounded by Yots.

Question 4.

A programming language uses the following syntax:

```
Digit = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
Letter = A | B | C | D | E | F
Variable = (<Letter> | <Digit>) {(<Letter> | <Digit>)}
PRINT = PRINT (<Digit> | <Letter> | <Variable>)
Assignment = LET <Variable> = (<Variable>|<Digit>|<Letter>)
Statement = <PRINT> | <Assignment>
Loop = FOR <Variable> = <Digit> To <Digit> {<Statement>} ENDFOR
```

(a) Draw a railroad diagram to show the syntax of an `Assignment`.

(b) Explain why `LET A = 75` is a valid `Assignment`.

(c) Write program code, using the syntax defined above, which will print the numbers from 1 to 5 inclusive.

Suggested Solutions

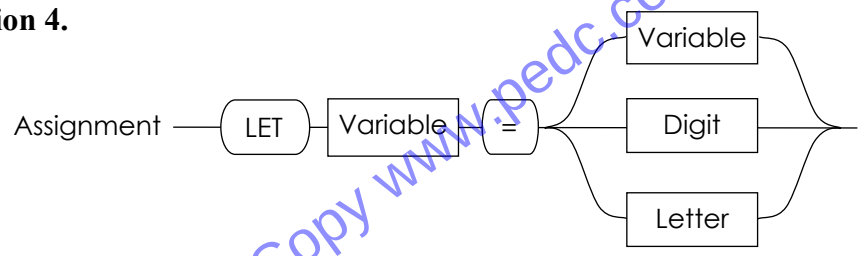
1. (D)

2. (B)

3. (C)

Question 4.

(a)



(b) A is a valid variable because A is a valid letter and a variable can be a single letter.

75 is a valid variable because 7 and 5 are both valid digits and a variable can be one or more digits.

Let A = 75 is a valid assignment because it is in the form LET variable = variable, and A and 75 are both valid variables as explained above.

(c)
```
FOR A = 1 To 5
    PRINT A
ENDFOR
```

SET 5A

1. A metalanguage is:
   (A) either an EBNF or railroad diagram.
   (B) used to make sense of programming languages.
   (C) another name for a high-level language.
   (D) used to describe the syntax of another language.

2. In EBNF {<thing>} means that <thing>:
   (A) is repeated one or more times.
   (B) is repeated zero or more times.
   (C) is optional.
   (D) can take on more than one value.

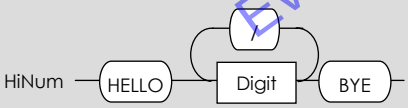
3. Which of the following is true of multi-dimensional arrays?
   (A) Data is stored in records.
   (B) Each data item takes the same value.
   (C) All data items are of the same data type.
   (D) No two data items can have the same value.

4. Declaring a user defined variable involves:
   (A) Defining the data type then declaring the variable.
   (B) Declaring the variable then defining the data type.
   (C) Determining the number of indexes then declaring the variable.
   (D) Opening the data type and then reading its contents.

5. If RND generates a random number in the range 0 to less than 1. INT(RND*6)+4 would result in integers in the range:
   (A) 4 to 10.
   (B) 6 to 10.
   (C) 4 to 9.
   (D) 3 to 9.

6. A multi-dimensional array requires 3 indexes and needs to store 1000 data items. Which Visual Basic statement achieves this task?
   (A) DIM Test(10,10,10)
   (B) DIM Test(9,9,9)
   (C) DIM Test(999,999,999)
   (D) DIM Test(400,300,300)

7. A data structure containing a series of records could be:
   (A) an array of records.
   (B) a file.
   (C) a multi-dimensional array.
   (D) Both A and B.

8. 

   If a digit is defined in EBNF to be 1|2|3|4|5. A syntactically correct HiNum is:
   (A) HELLO/1/2/1/BYE
   (B) HELLO1232/BYE
   (C) HELLO1/2/3/4BYE
   (D) HELLO1/2/3/BYE

9. A data structure is declared in Pascal using the following statements:

```
TYPE
  ThisType = RECORD
       This : Integer;
       That : Boolean
  END;
  ThisThat=ARRAY[1..3] OF ThisType;
```

   Which of the following is correct?
   (A) An array of records exists called ThisThat. It contains six data items.
   (B) Each record of type ThisThat will contain six data items.
   (C) A variable declared with type ThisThat will be an array with 3 records.
   (D) A variable of type ThisThat will contain two records. Each record containing three fields.

10. What is the purpose of the following fragment of Visual Basic code:

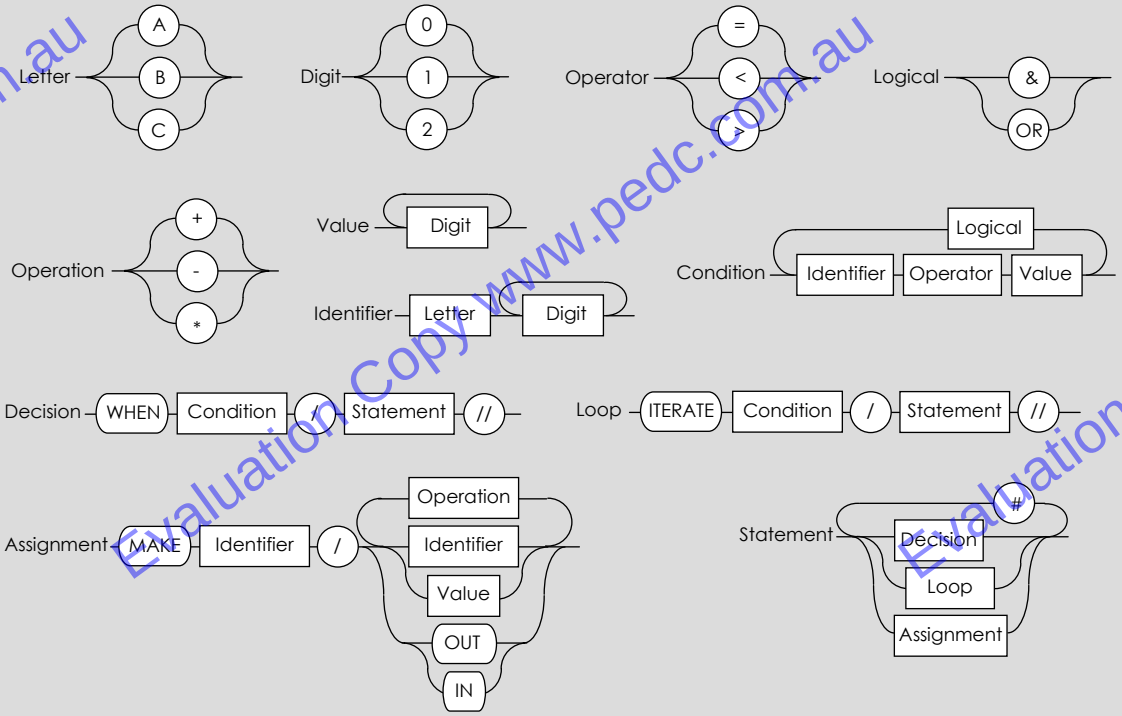
```
Count=1
While Count<=1000
   Result = rand.Next(0, 2)
   Total(Result) = Total(Result) + 1
   Count = Count + 1
End While
```

   (A) A loop counts from 1 to 1000. 1000 random numbers are totalled. Each number is either a 0 or a 1.
   (B) The number of 0s and the number of 1s generated is calculated for 1000 random numbers. Each random number can be either a 0 or a 1.
   (C) One thousand 0s and 1s are generated. The total of all the 0s is stored in Total(0) and the total of all the 1s in Total(1).
   (D) 1000 random numbers from 0 to 1 are generated. Each number is added to the total after it is generated.

11. The ability to store and retrieve data from secondary storage devices is vital to most applications. Explain the processes involved when writing source code to read from and write to disk files.

A new programming language called Zenith has been developed. Zenith uses a Base 3 number system for all its calculations, hence only the digits 0, 1 and 2 are required. In Base 3, we count 1, 2, 10, 11, 12, 100, 101, 102, 110, 111, 112, 120, 121…

The railroad diagrams below describe the syntax of this language:



12. Write down five legal Zenith identifiers.

13. Convert the Zenith assignment command into an EBNF diagram.

14. Zenith uses a multi-purpose assignment command. This command performs the usual assignment function together with input and output. Create a code fragment, in Zenith, that will get a number then print out all the numbers from 1 to that number.

15. Division is not included as a standard operation in Zenith. Write a code fragment in Zenith that performs division using repeated subtraction. You will need two outputs; the quotient and the remainder.

TRANSLATION METHODS IN SOFTWARE SOLUTIONS

Translation is the process of converting high-level code or source code into machine executable code. High-level languages cannot be understood directly by computers, they need to be converted to a form that is understood by the processor. This is similar to speaking in German to an English speaker. The German needs to be translated into English before the meaning can be understood.

Source code is said to be machine independent. This means it can be used on a number of different processors. In reality, some changes are normally required for this to occur. Executable code, on the other hand, is very processor specific. Each family of processors will have different machine language instructions. Source code needs to be translated for use on specific processors. If our German speaker wished to convey the same message to both an English speaker and an Italian speaker he would require two translators.

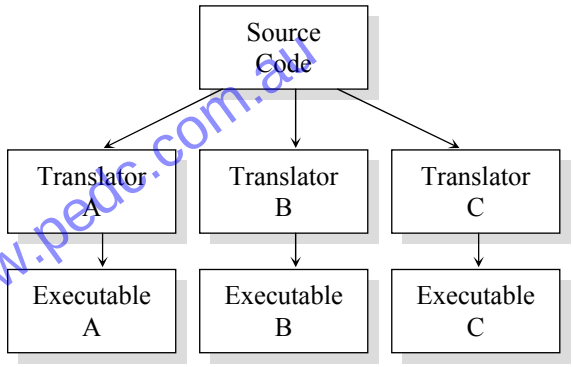


Fig 5.7
Different translators are used to create executable files for different processors using the same or similar source code.

The introduction of the World Wide Web has resulted in languages that can be implemented on a wide range of unknown processors. Languages such as Java, undergo a two-stage translation process. The source code is translated into byte code. This byte code is distributed and translated on the end-user's machine into executable code specific to the user's machine. Each end user machine has translation software installed to perform the byte code to machine code translation, in the case of Java this software is known as the Java Virtual Machine (JVM). A different JVM is installed for different operating systems and devices. This system can be likened to the symbols used to represent mathematical processes. A German mathematician writes his formulas using mathematical symbols, which are understood throughout the world. English and Italian speakers are able to translate these mathematical symbols themselves into their own language.

Let us now examine the two most common methods of translation:

- Interpretation
- Compilation

We will then examine the translation process in detail.

GROUP TASK Discussion
'A large variety of translators are available for the C++ language. Wouldn't it be better if we could just have one standard translator that all C++ programmers could use?'
Do you agree with this statement? Discuss.

GROUP TASK Research
Java executes on a broad range of devices. Using the internet or otherwise create a list of different devices which utilise a JVM to execute Java code.

Interpretation

Each line or statement of source code is translated into machine code and then immediately executed. If errors exist in the source code, they will cause a halt to execution once encountered by the interpreter. The actions of an interpreter are similar to that of a speech interpreter. If our German speaker wishes to have a conversation with an English speaker then the speech interpreter translates each sentence into English as soon as it is spoken in German. The English version is then spoken to the English-speaking recipient. If the speech interpreter cannot understand a German sentence, he is unable to translate it into English. The German sentence must be changed to a form the speech interpreter can understand before the correct translation can take place. Because the interpreter is there at the time the communication takes place, it is easy to make an on the spot change and then continue with the translation process. High-level language interpreters provide a similar advantage to the developer. Errors are generated as they occur and can be corrected. Translation and execution can then continue.

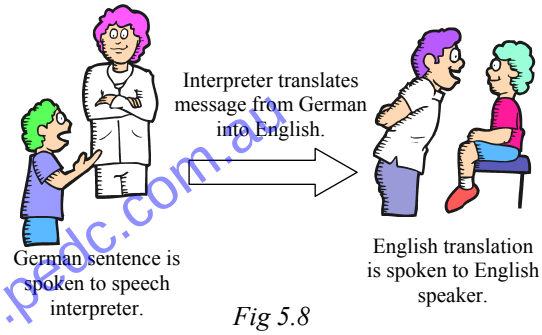


Fig 5.8
Speech translation can be likened to source code interpretation.

Interpretation does slow down the process of execution significantly. Each statement must be translated into executable code before it is executed. The translation process creates a significant time overhead. Because the interpreter only receives one high-level language statement at a time, it is not able to look ahead and translate each statement in the most efficient manner. Other methods of translation are able to optimise their output based on the overall view of the total source code.

Users of interpreted programs must have a copy of the interpreter installed on their machines for execution to take place. This requirement involves further costs and memory overheads for the users of interpreted software. Because the actual source code is distributed to users, the developer has limited control over his intellectual rights. Users can easily access the source code and use it for their own purposes.

Currently interpreted languages are routinely used on web servers to write server side scripts – PHP and Microsoft's ASP are common interpreted examples. When customising existing applications often the scripting language is interpreted.

For most compiled languages an interpreter is also available for use as a development tool by software developers. Many software development environments include an interpreter for use during coding which allows the programmer to execute small segments of code quickly as they are coding. The final distributed product is produced using a compiler.

GROUP TASK Activity
Interpreters translate each of line of code and then execute it if no errors are found. Create a flowchart to describe the operation of an interpreter.

GROUP TASK Research
Make a list of programming languages which are generally interpreted at runtime and briefly outline the main use of each language.

Compilation

The source code is translated into executable code. The executable code can later be executed without any need for the translator. Compilation is a batch process. The input to this process is the high-level source code and the output is the executable file or files. Errors encountered during the compilation process are normally relayed as a series of messages to the programmer. All coding errors are reported at the end of an unsuccessful compilation operation.

The operation of a compiler can be likened to a translator translating a book into a foreign language. Imagine a translator is converting a novel written in German into English. The translator will translate the entire text of the novel. Once the manuscript in English has been produced in its entirety it is printed and distributed. It is difficult for readers of the English version to realise the language used in the original German book. Let's take our analogy one step further. The translator will no doubt find various sentences in the original German difficult to translate into English. The translator would make a note of these sentences. At the end of the translation process, these sentences would be analysed and perhaps the original meaning would be acquired from the author. The translator then goes back over the manuscript correcting their translation. This is similar to a compiler listing all the errors found in the source code and the programmer going back and correcting these errors. There is one important difference however. A compiler does not produce any actual machine code until the entire source code is known to be correct in terms of the syntax of the source code.

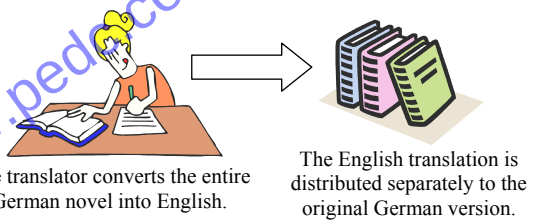


Fig 5.9
Compilation can be likened to the translation of a novel from German into English.

Compilers are used to produce executable code for the majority of commercial applications. Compiled programs generally run faster and more efficiently than similar interpreted products. As a compiled program is made up of machine code instructions, it is very difficult for the details of the original source code to be realised. This protects the intellectual rights of the author of the software. As the source code is not available to users, it is virtually impossible for them to make changes to the product. This makes it easier for software companies to provide quality support for their products, as they know precisely the workings of the application.

Executables created with compilers will always be machine specific. If a product is to operate on a different processor or operating system, then the source code will need to be recompiled using a compiler that produces the appropriate machine code instructions.

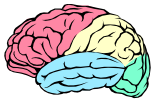
GROUP TASK Activity
Compilers translate the entire source code into an executable file or files (assuming no errors are found). These executables are run at some later stage separate to the compiler. Design a flowchart to describe the operation of a compiler.

GROUP TASK Research
Make a list of programming languages which are generally compiled and briefly outline the main use of each language.

Pages 243 to 263

Not included in this sample chapter

The techniques and software tools discussed in the next section are aimed at assisting the programmer to detect and identify the source of logic errors. Once identified, the programmer themself must correct the incorrect logic.



Consider the following:

A function is required that adds up counting numbers. For example, the call SumInts(4) should return the value 10 which is 1+2+3+4. The function executes with no errors reported, however the result returned is not correct. Clearly, the function contains one or more logic errors.

```
Function SumInts(SumTo:Integer):Integer;
VAR
Counter,Sum:Integer;
BEGIN
   Counter:=0;
   Sum:=0;
   WHILE Counter >=SumTo DO
      BEGIN
         Counter:=Counter+1;
         Counter:=Sum+Counter;
      END;
   SumInts:=SumTo
END;
```

Fig 5.26
Pascal function to add up counting numbers. (Including logic errors).

GROUP TASK Activity
Identify and correct the logic errors in the Pascal SumInts function shown in *Fig 5.26*.

Runtime Errors

Runtime errors are errors detected by the computer whilst a program is executing. There are many possible reasons for the system to generate a runtime error. The error, or bug, can be linked to either hardware or software problems. As we are primarily interested in software, let us consider the possible source of software runtime errors:

Bug
An error or defect in software or hardware that causes a program to malfunction. Apparently a real moth trapped between two relays caused the first computer bug.

- BIOS (Basic Input Output System)

The BIOS provides the interface between the operating system and the hardware. BIOS settings are stored in CMOS (Complementary Metal Oxide Semiconductor). CMOS is a type of RAM that requires very little power to retain its contents. A battery in the system box supplies power to the CMOS when the computer is turned off. Incorrect CMOS settings will often result in runtime errors.

- Operating system

Runtime errors generated by the operating system can result from various sources. Often BIOS, hardware driver or application software errors will cause an operating system runtime error. The all too familiar MS-Windows family blue screen is the result of an operating system runtime error.

Crash
A serious computer failure. The computer itself stops working or an application aborts unexpectedly. Caused by hardware malfunction or a serious software bug.

- Hardware drivers

Hardware drivers are normally loaded by the operating system. These are programs that provide an interface between the operating system and any hardware devices

installed on the computer. For example; hard disks, floppy disks, CD-ROM drives, sound cards, monitors, printers, scanners, modems, etc. The hardware driver itself can generate runtime errors, in which case a total system crash does not occur. If the problem is more substantial an operating system runtime error will occur, in which case it is likely the computer will crash.

- Application software

Runtime errors generated by application software can have a number of causes. Arithmetic overflow, division by zero errors and accessing inappropriate memory locations are three common possible causes.

An arithmetic overflow error occurs when a value is assigned to a variable that is outside the range allowed for that data type. For example, in many programming languages a variable declared as an Integer has a range from -32768 to 32767. Trying to assign a value outside this range to an identifier declared as an integer variable will result in an overflow error.

Division by zero is undefined in mathematics hence a runtime error is generated if this situation occurs. Often unexpected inputs result in division by zero errors. For example, calculating the average of a list of numbers will result in a division by zero error if the list is empty.

Accessing inappropriate memory locations occurs when an attempt is made to assign a value to an identifier that either does not exist or is of a different data type. For example, if an array called ThisArray is declared with a subscript range of 0 to 10, an attempt to assign a value to ThisArray(11) will cause a runtime error. Usually the error message will be something like 'Subscript out of range'. If we have a variable of integer type called MyInteger and another called MyString declared to hold characters then assigning MyString to MyInteger will result in a runtime error such as 'Type mismatch'.

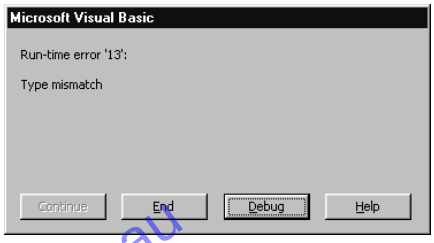


Fig 5.27
A runtime error message indicating a type mismatch has occurred.

Various other runtime errors are possible, many occurring as a result of a logic error in the code. For example, stack overflow errors often occur when a subroutine inadvertently makes a recursive call. A recursive call is when a subroutine calls itself. Each time a function is called an entry is placed onto the stack. Recursive calls can result in an attempt to place an infinite number of calls onto the stack resulting in a stack overflow runtime error.

Runtime errors resulting from the current project can often be anticipated. Most programming environments allow the program to intervene when a runtime error occurs. Exception handling is a common technique used by many languages. Subroutines should be written to anticipate and deal with runtime errors that may occur. Error subroutines should attempt to correct the problem; if this is not possible, they should attempt to save any data files before exiting.

GROUP TASK Investigation
Explore the help files in a programming environment with which you are familiar. Determine how the language deals with the detection and avoidance of runtime errors.

TECHNIQUES FOR DETECTING ERRORS IN CODE

Many techniques are available to the software developer to assist in the creation of error free source code. In this section, we examine a number of techniques commonly used during the implementation stage of the software development cycle. Each method aims to reduce the number of logic errors in the final code.

We will examine the following techniques:

- Stubs
- Drivers
- Flags
- Debugging output statements
- Peer checking
- Desk checking
- Structured walkthrough
- Use of expected output

Stubs

A stub is a small subroutine used in place of a yet to be coded subroutine. The use of stubs allows higher-level subroutines to be tested. Stubs do not perform any real processing; rather they aim to simulate the processing that will occur in the final subroutine they replace. Stubs are used to set the value of any variables affecting their calling routines and then end. Sometimes a stub may include an output statement to inform the programmer that the call to the stub has been successful.

The creation of stub subroutines is required when software projects are coded using a top-down design methodology. Because higher-level subroutines are created before lower-level subroutines, it is necessary to create 'dummy' subroutines. This enables testing of the higher-level subroutines whilst they are being coded.

Drivers

A driver, in general terms, provides an interface between two components. A driver controls the operation of some device. For example, a hardware driver is a program that provides the link between the operating system and a specific hardware device. Another example is a printer driver used to control the operation of a printer.

Fig 5.28
The driver of a car provides an interface between the environment and the car. The driver controls the operation of the car.

In terms of software development, a driver is a subroutine written to test the operation of one or more other subroutines. Commonly, drivers set the value of any required variables, call the subroutine to be tested and output the results of the subroutine call. In essence, the driver is an interface between the subroutine and the programmer. The driver controls the subroutine.

Drivers are required when software projects are coded using a bottom-up design methodology. Lower-level subroutines are developed before higher-level subroutines. Because of this, it is necessary to write a driver to test the operation of each subroutine as it is being coded.

Flags

A flag is used to indicate that a certain condition has been met. Usually a Flag is set to either true or false, that is, most flags are Boolean variables. We have examined the use of flags within algorithms in chapter 4 and then again in relation to the operation of the CPU earlier in this chapter. In each case, flags are used to signify the occurrence of a certain event.

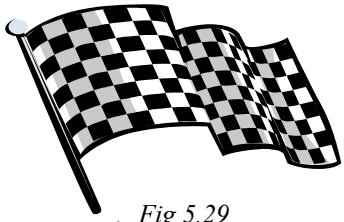


Fig 5.29
A black and white chequered flag is used to signify the end of a motor race. Either the race is over or it is not.

In terms of error detection, flags are used to check if certain sections of code have been executed or certain conditions have been met. For example, if a subroutine is called, a particular flag may be set to true. By examining the value of the flag, the programmer can determine the flow of control through the program. Setting the value of flag variables at different points in the code assists the programmer with the detection of the source of bugs.

Debugging output statements

Strategic placement of temporary output statements can help to isolate the source of errors. By placing output statements within called subroutines, the programmer can determine which subroutines have been called. This assists in the detection of the source of an error. Often a debugging output statement will be progressively moved as the debugging process continues. In this way, the flow of execution through the code can be precisely monitored. Eventually the source of the error is detected.

Often output statements will be used to display the contents of variables at crucial stages during execution. This allows the programmer to monitor changes in the contents of variables whilst the code executes. The precise place in the code where an erroneous calculation has occurred can then be determined.

Output statements are often used in conjunction with stubs, drivers and flags. In these cases, the output statement is used to indicate a particular event has occurred. A stub may contain an output statement to confirm it was actually called. Drivers usually contain statements to output the results of a call to a subroutine and the value of flags needs to be displayed during execution.

GROUP TASK Discussion
You are writing a function that sorts an array of data. Unfortunately all is not well, your subroutine is not working as expected. Stubs, drivers, flags and debugging output statements are all methods of error correction. Discuss how each of these techniques could be used to isolate the source of the error(s) in your sort function.

Peer Checking

Often errors will occur that seem to be impossible to correct for the original programmer. Colleagues on the development team are often able to see the problem from a fresh point of view. Peer checking involves other team members in the task of analysing each other's work to detect and correct errors.

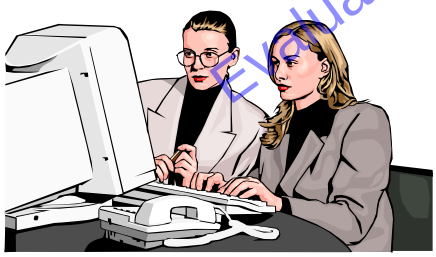


Fig 5.30
Informal peer checking is crucial to the successful development of software products.

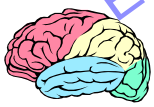
Many software development companies require that each subroutine is 'peer checked' as part of their quality assurance strategy. Errors that go unnoticed or uncorrected and are released to the public can cost software companies both in terms of direct financial costs as well as customer satisfaction. Formal peer checking involves programmers formally signing each other's work to verify they have thoroughly checked each coded subroutine.

Informal peer checking is used by virtually all programming teams. The flow of comments and ideas between team members is crucial to the successful development of any software product. This is particularly true in the case of detecting and correcting errors.

Desk checking

Desk checking is the process of working through an algorithm or piece of source code by hand. A table with a column for each variable is used. As the algorithm or code is worked through by hand, changes to variables are made by writing the new value under the appropriate identifier.

A desk check is particularly useful when the workings of a piece of source code are not fully understood. The process of working through the code statement by statement helps to make the logic clear. Often errors in the code will be detected as the desk check progresses. These errors can then be corrected and a new desk check commenced.



Consider the following:

A function is under development that will perform a linear search for a particular element amongst an array of elements. The function returns the value true if the item is in the list and false if it is not. The function has been written in Visual Basic but contains errors. *Fig 5.31* shows the function in its current state.

The programmer decides to perform a desk check of their code using the following test data array.

```
Animal(0) = "Frog"
Animal(1) = "Dog"
Animal(2) = "Cat"
Animal(3) = "Cow"
Animal(4) = "Ant"
Animal(5) = "Moose"
```

They decide to perform the desk check four times using the following four calls:

```
Linear(Animals(),"Frog")
Linear(Animals(),"Cat")
Linear(Animals(),"Moose")
Linear(Animals(),"Bat")
```

```
Public Function Linear(Search(), FindItem) As Boolean
Dim Lower, Upper, Count As Long
Dim Found As Boolean
Found = True
Lower = LBound(Search)
Upper = UBound(Search)
Count = Lower
While Count < Upper
   If Search(Count) = SearchItem Then
      Found = True
      Count = Upper
   End If
   Count = Count + 1
End While
Return Found
End Function
```

Fig 5.31
Linear search Visual Basic function with errors.

The desk check for the Linear(Animals(),"Frog") call is reproduced below:

Found	FindItem	Lower	Upper	Count	Search(Count)
True	Frog	0	5	0	Frog
True				5	
				6	

The code correctly finds Frog and returns the value true.

Let us now perform the desk check with the second test call Linear(Animals(),"Cat"):

Found	FindItem	Lower	Upper	Count	Search(Count)
True	Cat	0	5	0	Frog
				1	Dog
				2	Cat
True				5	
				6	

Again, the call correctly returns the value true.

GROUP TASK Activity

Complete the desk check with the remaining two calls:
Linear(Animals(),"Moose") and Linear(Animals(),"Bat")
There are actually 2 errors in this function. What are these errors and how can they be corrected?

Structured walk through

Structured walk throughs, as the name suggests, are more formal than peer checks. The developer or team of developers present their work to a group of interested parties. This group may include representatives from management, marketing and potential users, with the aim being to present the software and formally work through its functionality. The developers walk the group step-by-step through each aspect of the program. As the walk through continues, comments are written down for future consideration. No attempt is made to correct or justify aspects of the product; the aim being to receive feedback on the product as it stands.

Structured walk throughs are normally undertaken as formal meetings. Each person in attendance at a structured walk through should be given all relevant documentation prior to the meeting. This allows them to have an overall view and feel for the product and its design. The walk through itself should be a demonstration of the product and its design. Comments can either be made verbally or may be restricted to written comments. In either case, a response to comments should not occur during the walk through, rather at a later time when they can be considered carefully.

Fig 5.32
Structured walk throughs are normally undertaken as formal meetings.

Structured walk throughs can be used to evaluate the design at different levels. Their aim is to explain in a structured manner the operation of some part of the design and development process and to obtain feedback from interested parties. They may be

used to walk through the logic within an algorithm, the source code written for a specific module or they may be used to evaluate the final product once it has been implemented in code. In fact, most aspects of the software design and development process can be evaluated using structured walkthroughs.

GROUP TASK Discussion
Why do you think comments and suggestions should be collected but not responded to until after the walk through is complete? Surely it would be better to discuss any issues during the walk through. Discuss.

Use of expected output

When initial test data items are created the expected output from these inputs should be calculated by hand. Once the subroutine has been coded, the test data is entered and the output from the subroutine is compared to the expected outputs.

If the actual and expected output match then this provides a good indication that the subroutine is performing accurately. If they do not match then clearly a logic error has occurred. Further techniques must be employed to determine the source of the error.

If the test data has been designed to test every path through the source code and every boundary condition then a 100% match between actual and expected output means the source code is logically correct. It does not necessarily mean the source code is efficient, high quality or free of runtime errors.

For large programs, it is impractical to test every path through the entire program. A typical application can contain many thousands of lines of code. The number of test data sets easily runs into the billions. If subroutines have been designed using the 'one logical task per subroutine' rule, then it should be possible to perform the test on each subroutine.

Currently test driven development (TDD) methodologies are becoming popular. When TDD is used then thorough tests (together with test data) are written prior to coding. Passing these tests indicates the source code is achieving its purpose. As new subroutines are coded new tests are added, all these tests can be rerun each time new or modified code is added to the project. Often the source code for the test routines can be as large as the source code for the actual project.

GROUP TASK Activity
'If every path and every boundary condition has been tested and the output matches exactly the expected output then surely the code must be free of errors.' This statement is not true. Can you give examples of errors in code that will not be found by testing each path and boundary?

GROUP TASK Research
Many universities now teach computer science using a test driven development (TDD) methodology. Research using the internet or otherwise to find examples of teaching resources used by computer science departments in universities which demonstrate TDD.

SOFTWARE TOOLS FOR DETECTING ERRORS IN CODE

Most software development environments include tools to assist the programmer in the task of isolating errors in the source code. The majority of environments include automatic syntax checking as statements are entered. Different environments will provide different tools to assist in the identification and isolation of logic and runtime errors.

The following five tools are commonly available to assist in the detection of logic and runtime errors:

- Breakpoints
- Resetting variable contents
- Program traces
- Single line stepping
- Watch expressions

Breakpoints

Breakpoints are used to temporarily halt the execution of the code. Once execution has stopped it is possible to examine the current value of each variable. By adding breakpoints at strategic points within the code, it is possible to locate the source of the error.

Setting breakpoints is performed in one of two ways. Many environments provide a 'break' statement, which is entered in the code where a break is desired. Other environments provide a menu item that inserts a breakpoint at the current cursor position.

In Visual Basic, once a break point is reached execution halts (see *Fig 5.33*). By holding the mouse over identifiers in the source code, the current value held in the variable is displayed. An immediate window allows for code to be entered for instant execution.

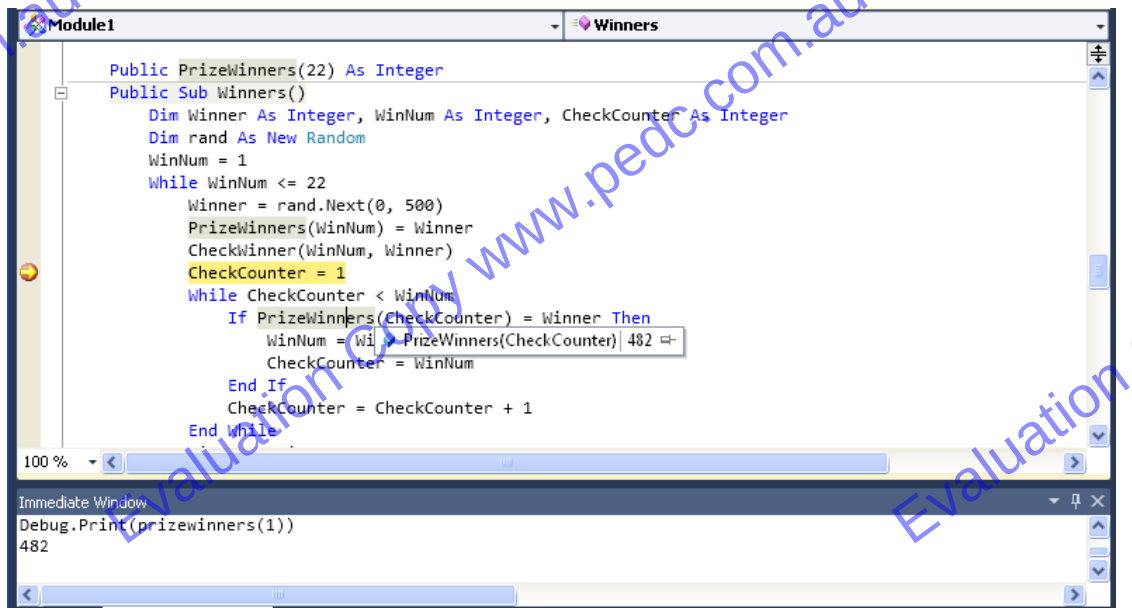


Fig 5.33
The Visual Basic (VB.NET) environment in break mode.

Resetting variable contents

Altering the value of variables during the execution of code can be useful in determining the nature of an error. The ability to alter a variable's contents can also allow the programmer to determine which values are causing errors without the need for the variable to actually attain that value as a consequence of execution.

Often the contents of a variable will be changed whilst execution has been halted by a breakpoint. For instance, in *Fig 5.33* we see that PrizeWinners(1) = 482. In the immediate window we could enter an assignment statement to alter the value of PrizeWinners(1) to a different value to test a particular scenario.

Inserting assignment statements within the source code itself can also be used to temporarily alter the contents of variables. For example, if the programmer wishes to ensure a particular piece of code is executed then inserting a temporary assignment statement can force execution to progress in a particular direction. Perhaps a particular calculation is causing problems, variables included in the calculation can be reset to differing values to allow the programmer to thoroughly analyse the calculation.

Program traces

Tracing, in terms of error detection, refers to tracking some aspect of the software. A program trace enables the order of execution of statements to be tracked or the changing contents of variables to be tracked. Both these types of trace involve analysing and following the flow of execution.

Various forms of trace are possible. Some programming environments can display each line of code as it is executed at slow speed. Others allow you to analyse the call stack when a breakpoint is encountered. The call stack is a list of subroutine names from the current call back to the main program. Many COTS products, particularly data oriented products, are able to produce a trace file. This file maintains a log of all the transactions that have taken place. This trace log can be analysed to identify the source of an error.

Many development environments provide a window containing all the variables local to the current subroutine together with their current value and data type. This list is updated each time the contents of a variable changes. This window is used in conjunction with breakpoints or single line stepping. Examining the changing contents of variables can assist in tracing the source of errors.

GROUP TASK Discussion

Division by zero and overflow runtime errors are highlighted by the system where they occur. Often the calculation that ultimately caused the runtime error is not at fault. Discuss methods of identifying the true source of these errors using breakpoints, resetting variable contents and program traces.

Single line stepping

Single line stepping is the process of halting execution after each statement is executed. Each statement is highlighted as it is executed. A simple keystroke allows execution to continue to the next statement. For example, in VB.NET the F8 key steps through the code line by line.

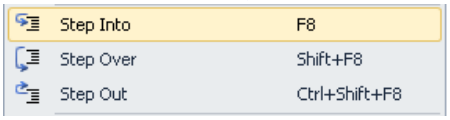


Fig 5.34
Stepping tools in VB.NET

Single line stepping can become somewhat monotonous when a large number of procedure calls or loops are involved. Most environments provide variations on the single line stepping idea. Commands to allow execution to step over procedure and function calls assist in isolating an error to a particular subroutine. Step out options mean that procedure and function calls are completed before execution once again halts. This style of stepping allows the programmer to concentrate on code within the higher-level subroutine.

Watch expressions

A watch expression is an expression whose value is updated in the watch window as execution continues. Usually watch expressions are variable names. In *Fig 5.35* the variable CheckCounter, Winner and PrizeWinners have been entered as watch expressions. Watch expressions can also be calculations such as Price + GST.

By combining single line stepping and watch expressions, an automated desk checking system can be created that can be used to watch the value of particular expressions. The watch window is similar to an individual row of a manual desk check.

Watch

Name	Value	Type
CheckCounter	4	Integer
Winner	54	Integer
PrizeWinners	{Length=23}	Integer()
(0)	0	Integer
(1)	234	Integer
(2)	70	Integer
(3)	216	Integer
(4)	490	Integer
(5)	28	Integer
(6)	166	Integer
(7)	232	Integer
(8)	54	Integer
(9)	0	Integer
(10)	0	Integer
(11)	0	Integer
(12)	0	Integer
(13)	0	Integer
(14)	0	Integer
(15)	0	Integer
(16)	0	Integer
(17)	0	Integer
(18)	0	Integer
(19)	0	Integer
(20)	0	Integer
(21)	0	Integer
(22)	0	Integer

Fig 5.35
Watch window in VB.NET displaying the current value of watched variables and expressions as the code runs.

GROUP TASK Discussion

A subroutine is under development that reads a sequential file of records. As each record is read various calculations are made. A logic error exists, as some of the calculations on particular records are incorrect. Explain how the use of single line stepping and watch expressions could be used to help isolate the problem.

GROUP TASK Activity

Use a programming environment with which you are familiar to perform the following: Write a short program to find the average of 10 random numbers between 1 and 100 inclusive. Create watch expressions for each of the variables in your code. Step through the code line by line to ensure it is operating correctly.

HSC style questions:

1. A new software application has been installed. A number of users report that the result of a calculation performed by the application is incorrect. What type of error has occurred?

(A) Logical
(B) Syntax
(C) Runtime
(D) User

Refer to the following algorithm when answering Question 2 and 3. Line numbers are included to assist when answering question 3.

```
10  BEGIN
20      numberOfStudents = 0
30      totalStudentMarks = 0
40      INPUT studentMark
50      WHILE studentMark >=  0
60          totalStudentMarks = totalStudentMarks + studentMark
70          numberOfStudents = numberOfStudents + 1
80          INPUT studentMark
90      ENDWHILE
100     averageMark = totalStudentMarks / numberOfStudents
110     PRINT averageMark
120 END
```

2. If the above algorithm is implemented and the first mark entered is negative then an error occurs. What type of error is this?

(A) Arithmetic overflow error
(B) Division by zero error
(C) Logic error
(D) Syntax error

3. Which of the following additional lines best corrects the "first mark entered is negative" error?

(A)
```
95  numberOfStudents = numberOfStudents + 1
```

(B)
```
95  IF numberOfStudents <> 0 THEN
115 ENDIF
```

(C)
```
95  IF studentMark < 0 THEN
115 ENDIF
```

(D)
```
93  IF numberOfStudents = 0 THEN
94      numberOfStudents = 1
95  ENDIF
```

Question 4.

As part of a software company's ongoing quality assurance procedures the following rules are followed by all programmers as they code subroutines.

- Global variables are not to be used.
- Subroutines must perform a single logical task.
- Completed subroutines must be peer checked prior to inclusion.

Explain why these rules will likely improve the quality of the company's products.

Suggested Solutions

1. (A) The code operates without displaying an error hence it cannot be a runtime or syntax error. The error is in the logic underlying the calculation.

2. (B) The body of the loop is never executed hence numberOfStudents is 0 when the division is attempted in line 100.

3. (B) Answer (B) effectively jumps the calculation and display of the average (lines 100 and 110) when there are no students. Although (D) also solves the problem it is not mathematically correct to display an average of zero when there are no student marks entered.

Question 4.

The widespread use of Global variables means that the use of the same variable names as a local variable will cause conflict and developers can never be sure where global values have been changed. If no global variables are allowed at all, then all data that needs to be shared between routines will need to be passed as parameters between the routines concerned. In this way, the different programmers involved in the development of the software have much more flexibility in their choice of variable names without fear of ambiguity or conflict arising when the various modules are pulled together into the final product.

When each subroutine performs one logical task only and is named appropriately to reflect that task, it is much easier to locate the source of logic errors during the testing phase. It is also easier for peer checking as the logic is much easier to follow by a programmer who is not the person who originally wrote it. In addition, future maintenance programmers can be more confident when they improve the processing of a routine as the job it performs is a single clear task.

The requirement to have each routine checked by another programmer prior to including it into the final version of the code helps ensure the code is error free, as well as ensuring that the source code adheres to the standards expected by the company. It also means that the final released version of the source code is more coherent, without each subroutine reflecting the individual preferences of the developer who wrote it. This makes it much easier to maintain such code, whether it is to customise it for a particular company or to remove errors found later.

GROUP TASK Discussion

Question 4 lists three rules. Suggest other possible rules or guidelines that would also assist a software company's ongoing quality assurance.

SET 5D

1. Many techniques are available to reduce the number of errors in the final solution. The most important technique is:
   (A) to use a clear modular structure.
   (B) use of debugging output statements.
   (C) to use drivers and stubs.
   (D) peer and desk checking.

2. Which of the following is true of logic errors?
   (A) They are discovered during translation.
   (B) They are usually discovered at runtime.
   (C) They always cause a halt in execution.
   (D) They result from misspelt reserved words.

3. The difference between peer checking and desk checking is:
   (A) Peer checking is the process of working through the code by hand. Desk checking is performed automatically by the computer.
   (B) Desk checking is the process of working through the code by hand. Peer checking involves analysing and checking code written by others.
   (C) Peer checking is used by large software developers to ensure code is error free. Desk checking is used to ensure algorithms are logically correct.
   (D) Desk checking and peer checking are different names for essentially the same process.

4. Constants used throughout a program should be assigned to an identifier. Why is this the case?
   (A) The value need only be changed in one place by future maintenance programmers.
   (B) The code executes faster.
   (C) Calculations will be easier to understand.
   (D) All of the above.

5. Which of the following is true of syntax errors?
   (A) They are always discovered during translation.
   (B) They are always discovered at runtime.
   (C) Can result in unexpected outputs.
   (D) They always result from misspelt reserved words.

6. Runtime errors are detected whilst the program is running. They can be the result of:
   (A) hardware faults.
   (B) faults in the operating system, BIOS or hardware drivers.
   (C) application software errors.
   (D) All of the above.

7. If a program is coded using a top-down design methodology, which of the following would be required to test each module?
   (A) flags.
   (B) stubs.
   (C) drivers.
   (D) break points.

8. Quality solutions:
   (A) perform the stated task without error.
   (B) will solve a single logical task.
   (C) are superior in efficiency, maintainability and design.
   (D) provide superior documentation to other solutions.

9. A temporary routine that calls other subroutines to assist in their testing is called a:
   (A) stub.
   (B) driver.
   (C) flag.
   (D) main program.

10. Temporarily halting the execution of code so that the contents of variables can be examined and altered is accomplished using:
    (A) watch expressions.
    (B) single line stepping.
    (C) program traces.
    (D) break points.

11. 'Coding of a software solution can be done from the top down or from the bottom up.' What does this mean? What techniques are required when coding using each of these methodologies?

The Pascal code fragment that follows, attempts to find all the factors of a number. Factors are numbers that divide evenly into a number. For example, the factors of 8 are 1, 2, 4 and 8. The TRUNC function returns the integer part of a number. This code fragment contains a number of errors.

```
Readln (Number);
Count := 0;
WHILE Count < Number DO
 BEGIN
   IF Number/Count = TRUNC(Number/Count)
     THEN Writeln (Number/Count);
   Count := Count + 1
 END;
```

12. Perform a desk check of the above code using 12 as the input. What errors are encountered? Suggest ways of correcting these errors. Perform the desk check again with your corrected code.

13. The above problem can be solved in a much more elegant and efficient manner. Make changes to the code to make it more elegant and efficient.

The modules for a large program have been developed separately. Apparently, all subroutines in each module has been thoroughly tested on its own. These modules are now being combined to form the total solution. A number of errors are occurring when the code is executed.

14. List and describe the types of errors that could be occurring. Give an example of each possibility.

15. For each of the error types identified in question 14, describe a possible technique for isolating the error.

DOCUMENTATION OF A SOFTWARE SOLUTION

Throughout the development of a software solution, various forms of documentation are created. We have already discussed the need for continually updating documentation throughout the development process. In this section, we examine the total documentation required before software solutions are released to clients.

Documentation can be split into two broad categories: user documentation and technical documentation. These categories describe the intended audience of the documentation. The nature of the software product and its intended audience will determine the language and complexity of each documentation item.

USER DOCUMENTATION

Users are the final operators of the product. Documentation targeted at users needs to be directed at their level of knowledge and expertise. A game designed for use by children will require very different user documentation to a product designed to automate the operations of a large corporation. The end users of the system must be identified before the production of user documentation commences.

There are many types of user documentation possible. In this section, we examine four specific types of documentation that are provided with the majority of software products. Each of these documentation types is commonly provided in either printed or electronic formats.

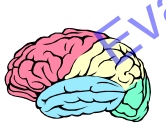
Installation Guide

Installation guides should provide the user with sufficient information to successfully install the software product on their machine(s). Hardware and software requirements will be included together with step-by-step instructions to guide the user through the set-up process. Often notes are included to help resolve common installation problems.

Hardware requirements should be clearly stated. Often a minimum set of hardware requirements will be given together with a recommended set of requirements. Many products require other software to be installed on the system prior to installation, operating systems being the most obvious example.

The main section of the installation guide describes the steps that need to be undertaken to install the product. For small applications, this can be as simple as placing a CD-ROM in a drive and waiting for the installation program to auto-run. Large systems often require significant direction from the user, particularly in regard to configuring the system for the particular installation site. For large systems an installer from the software company, or one of its distributors, often performs or directs the installation process.

Whilst is should not be expected, it is not uncommon to encounter errors during the installation process. The large variety of hardware and software combinations available cannot all be tested. As a consequence, installation errors can occur. The installation guide should attempt to address the most common of these. A good installation guide will also provide support contact details in the event that the user cannot resolve an installation problem.



Consider the following:

A chain of restaurants is installing a computer system to improve the productivity of their processes. The software has been written by HospSys, a software development company specialising in software solutions for the hospitality industry. The restaurants will be connected to their head office via dedicated modem connections. The software has been written and an installation guide is currently being prepared. An installer employed by HospSys will complete the actual installation of the software. The installer will also provide the initial training to users.

There are two audiences involved in this installation: the restaurant personnel and the installers from the software company. Specific sections of the installation guide are directed at each of these two groups. The installer will have a copy of the entire installation guide whereas the restaurant personnel require items pertinent to their particular installation tasks.

Let us examine items that are included in the restaurant personnel section of the installation guide:

Hardware and software requirements

- File server running Windows Server detailing minimum disk and RAM requirements.
- Workstations for order entry
- EFTPOS terminal with dedicated phone line
- Manager's workstation and printer
- POS (Point of Sale) terminal with receipt printer
- Network hub and associated cabling
- Modem with dedicated phone line

Installation steps

1. Order/Receive hardware and software from external vendor.
2. Schedule cable installation for network.
3. Schedule hardware/network installation.
4. Organise and install two phone lines. One near file server and one beside POS terminal.
5. Liaise with bank and have EFTPOS terminal installed.
6. Inform HospSys and schedule an installation date. Complete application software installation. Schedule employees of restaurant for initial training. Training completed by HospSys installer. System considered Live.

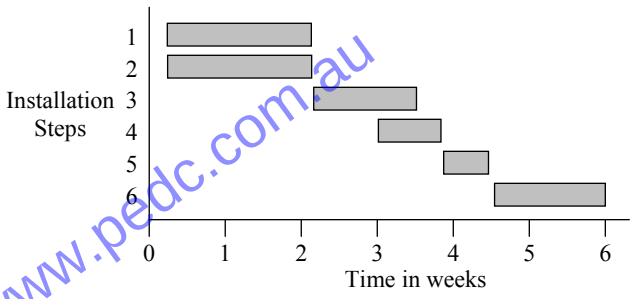


Fig 5.36
Gantt chart for the installation of the HospSys product.

GROUP TASK Discussion

The HospSys system is a client-server application. Each evening data from each restaurant in the chain is uploaded to head office. As part of the installation process the installer must ensure all aspects of the system are operational. Create a Gantt chart that could be included in the installer's section of the installation guide.

User Manual

Quality user manuals aim to provide concise and accurate information in regard to the operation and purpose of software. Topics in user manuals describe what tasks the software can complete, why these tasks are required and how these tasks are completed using the software. The information needs to be presented using a format that is appealing, accessible and consistent. Often much of the information traditionally contained in a user manual will be included in help files that are accessible directly from within the application.

User manuals can be organised in a number of ways depending on the nature of the software product. Topics can be ordered in terms of difficulty; in this way, users working through the manual progressively increase their skills. Topics can be presented in order of usage. Commonly used task are described first, with less common tasks described in the latter part of the manual. Software products containing a large number of tasks can be arranged in terms of function. For example, a user manual for an accounting package may contain topics on general ledger, accounts receivable and inventory and purchasing. Each topic describes the functionality of that particular aspect of the package.

Each topic within the user manual should provide two explanations of each of the software's functions; a conceptual explanation and a procedural explanation. Conceptual explanations explain what the function is and why the user may need to use that function. For example, a topic describing a save function explains what and why saving is necessary, for instance; 'Saving stores a copy of the file in a permanent location. Saving ensures the file is not lost if a power failure or runtime error occurs.' Procedural explanations describe how a process is performed using the software. For example, 'To save a file open the file menu and click on save.' Screen dumps are often used to assist with procedural explanations (see *Fig 5.37*).

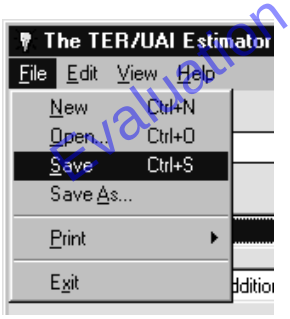


Fig 5.37
A screen dump displaying the Save command.

Reference Manual

Reference manuals are designed to be an efficient source of information. In terms of software, a reference manual should succinctly describe each command within the application. Reference manuals are not designed to be read from start to finish, rather they will be read in random order as needs arise.

Each command within the application should be listed in alphabetical order. This assists the user in locating required information without the need to refer to the contents or index pages. Normally each command is listed together with a brief description and a succinct example to illustrate its usage.

Often a quick reference card is included in the user's documentation package. This card lists all the commonly used commands together with shortcut keys and a brief explanation.

The ATAR Estimator
Quick Reference Guide

Menu Command	Shortcut Keys	Function
File – New	Ctrl-N	Create a new file
File - Open	Ctrl-O	Open an existing file
File – Save	Ctrl-S	Save the current file
File – Print – Individual	Ctrl-P	Print the current students results
Edit – Delete	Shift-Delete	Delete the current student record
Help – Using	F1	Open help
View – Individual	F2	Switch to individual student view
View – All Students	F3	Switch to all student view
View – By Course	F4	View students in a particular course

Fig 5.38
A quick reference card, showing commonly used commands and shortcut keys.

Tutorials

Tutorials provide instruction in the use of software products using example scenarios. The user is lead step-by-step through the processes included in the product. The user performs the tasks under the direction of the tutorial. Tutorials are designed so users can experience real world use of the application before using their own data.

Often sample data files are provided with applications for use during tutorials. These sample files are used to familiarise users with the functions available in the software product without the risk of damaging the real data. The use of sample files means that users do not need to input vast amounts of data to experience real life scenarios.

Tutorials should be split into individual lessons. Each lesson concentrates on developing a particular set of skills. Users can work through the tutorial at their own pace and as time constraints allow. Particular lessons can be undertaken, as the user requires the skills.

Paper-based tutorials, although still common, are being replaced by electronic versions. Usually a small window on the screen is used to direct the user through the tutorial.

GROUP TASK Investigation

User manuals, reference manuals and tutorials are the primary methods of providing instruction and information to users. Examine the written documentation included in various commercial software packages. Evaluate each item in terms of its intended function. How could these items be improved?

Online Help

Each of the above types of user documentation can be in either printed or electronic online form. It is now common for most user documentation to be provided online rather than as printed manuals. Online documentation can be provided as Adobe PDF files which are often similar in structure to more traditional printed manuals; however dedicated hypertext documents are now more common. Hypertext help documents allow users to efficiently search for specific items or in many cases they allow context sensitive help to be provided from within the application. When the user selects help within the application they are directed to the most relevant help topic automatically.

Online help can be truly online, meaning it is stored on a remote web server and hence requires an internet connection or it can be stored on locally on the user's machine. Currently many software applications only provide truly online internet based help. This has the advantage that the software company can update the help files as required and all users will immediately have access to the new content. In addition, interactive help such as user support forums and the ability to ask questions or suggest new functionality are valuable additions to an applications support options.

GROUP TASK Research

Consider a popular software application installed on your computer. Research and briefly describe all the various types of online help available to users of this software application.

TECHNICAL DOCUMENTATION

Technical documentation is designed for an audience who are proficient and knowledgeable in regard to the subject matter. In terms of software products, technical documentation primarily describes the structure and engineering behind a product.

Whilst software is being developed, thorough documentation should be maintained. These documents form the basis of the final technical documentation. Personnel involved in maintaining, upgrading and supporting software solutions make use of technical documentation.

Log Books (or process diaries)

A log book records the systematic series of actions that have occurred during the development of a project. Log books or process diaries are often utilised during the design and development of products in many industries. Maintaining a log book is a method of chronologically recording the processes undertaken to develop a final product. Individual members of the development team can maintain their own log books or a central log book can be maintained for each product under development.

The purpose of maintaining log books is twofold. Firstly, the information can be used to determine the relative cost of each stage of development. More importantly, log books are used as reflective devices. By examining the log books mistakes, delays or inefficiencies during development can be analysed. Future projects can then be more effectively managed to ensure such issues do not continue to arise.

Consider the following:

The following log book description is from the *Software Design and Development Stage 6 – Software and Course Specifications* document available on the NSW BOS website. This description refers to a log book you might use whilst working on a project as part of the SDD HSC course.

Log books

Log books are used to document the progress of a project. Entries should include:

- date
- description of the progress (or lack thereof) made since the last entry
- tasks achieved
- descriptions of stumbling blocks or issues encountered and how they were managed
- details of possible approaches for upcoming tasks
- reflective comments
- reference to resources used

Log books may be produced using spreadsheets, blogs, handwritten entries or electronic journal entries. Email messages to a fellow developer may be useful, as they contain time and date stamps. A sample log book entry may look like:

April 22nd 2010 – problems with images I am so pleased the coding for my mainline is finally done! This morning I spent some time importing the pictures for each screenshot. I had some trouble with it. The one problem I still have to fix is the transparency so I will try to import each in GIF format and make the background transparent. Hopefully it will work.

Note that the comment is reflective, describes what has been achieved, identifies a particular stumbling block and possible approaches to dealing with it.

Fig 5.39
Software Design and Development Stage 6 – Software and Course Specifications p22

GROUP TASK Discussion

Discuss how the regular use of a log book would assist in the development of software projects within the SDD HSC course.

Source Code Documentation

The source code itself is probably the most important form of technical documentation. It is virtually impossible to isolate logic errors in an application without access to the original source code. For source code to be a valuable technical resource for future maintainers, it must be intelligently documented.

Documentation within the source code is often called internal documentation. Internal documentation can take two forms, comments and intrinsic documentation. Let us examine each of these in turn.

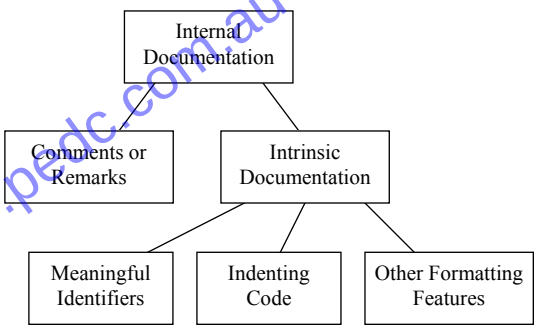


Fig 5.40

Source code documentation can take many forms.

- Comments

Comments provide information for future programmers. The translator ignores comments, or remark statements, within the code. Each procedure, function and logical process within the code should be preceded by a comment.

Comments should explain what a section of code does rather than how it does it. For example 'Find the largest value in the array' is a better comment than 'Make the first array value the largest then loop around for each remaining array item checking it against the current largest. If a value is larger it becomes the current largest value.' The second version of this comment is really echoing the code. Comments should summarise the code, enabling future programmers to grasp the intention at a glance. *Fig 5.41* is a Visual Basic 6.0 procedure. Reading the comments in this procedure explains clearly what the procedure accomplishes. The code itself provides information about how the processing is accomplished.

```
Private Sub RedGreenBlue(ColValue, Red, Green, Blue)

'ColValue is the input and is the 24 bit colour value.
'This procedure extracts the 0-255 value for each of
'the Red Green and Blue components and returns them
'in the parameters of the same name.

Dim HexCol

'First we convert the colour value to hexadecimal.
'The hex function returns a string. e.g. C05F3A.

HexCol = Hex(ColValue)

'Extract each colour value from the hex string.

Blue = Val("&H" & (Left(HexCol, 2)))
Green = Val("&H" & (Mid(HexCol, 3, 2)))
Red = Val("&H" & (Mid(HexCol, 5, 6)))

End Sub
```

Fig 5.41

Procedure containing comments describing the procedure and its logical tasks.

Comments that explain the logic of code should be used sparingly. Only for particularly difficult logic is a full explanation of the code's logic necessary. The code itself describes the logic. After all, our audience are programmers whom we presume can read code. Comments that explain the logic in detail can easily become incorrect as the code is modified.

GROUP TASK Investigation

In Basic a single quote is used to indicate a comment. In C++ two forward slashes are used. Examine a number of programming languages to determine how comments are indicated in the code.

- Intrinsic documentation

Intrinsic means belonging by its very nature. Intrinsic documentation is therefore documentation that is part of the code by its very nature. In other words, the code is self-documenting.

There are two main types of intrinsic documentation: the use of meaningful identifiers and indenting of code. Self-documenting code is easier for humans to read. The translator does not care what names are given to identifiers, just as long as they are syntactically legal. Translators ignore indenting.

```
PROCEDURE A; VAR B : Integer;
BEGIN B:=1; WHILE B<=10 DO BEGIN
WriteIn (Count); B:= B+1 END END;

PROCEDURE OneToTen;
VAR
   Count : Integer;
BEGIN
   Count:=1;
   WHILE Count<=10 DO
      BEGIN
         WriteIn (Count);
         Count:= Count+1
      END
END;
```

Fig 5.42
Intrinsic documentation makes source code understandable for humans.

An identifier is the name given to a variable, procedure, function or instance of an object. The design specification for the project will normally describe the naming conventions to be used within the project. In this text, we have generally used mixed case for identifiers. For example, in *Fig 5.42* the procedure is called OneToTen or could have been called oneToTen. Some developers use the underscore character to separate words, for example One_To_Ten. It is common practice to use upper case for all constant identifiers.

Identifiers should describe the purpose of the element it represents. If a variable is used to store client's names and addresses then appropriate identifiers could be: ClientDetails, ClientInfo or perhaps just Clients. Procedure and function names should describe the processing that occurs within the subroutine. For example, ExtractDate or SortArray.

Many programming languages have their own naming conventions. In Visual Basic, control names should begin with a mnemonic that identifies the particular type of control. Text box identifiers commence with txt, for example txtDate. Labels commence with lbl, for example lblResponse. List boxes with lst, combination boxes with cbo and so on. Adhering to these conventions enhances the readability of the source code.

Indenting is the process of setting lines of code back from the left margin. Indenting sections of code within control structures improves the readability of code. Many programming environments include automatic indentation settings. Consistent use of indenting allows the structure of the code to be seen at a glance. The beginning and end of each control structure stands out, this assists in visually describing the logic. The rules of pseudocode insist on the use of indentation, similar rules should be implemented within source code.

Many other formatting features are commonly used to improve the intrinsic readability of code. Leaving blank lines between comments and the code they describe. Colour can be used to visually differentiate between comments, reserved words, operators and operands. Ensuring each line of code can be viewed without the need for scrolling. Most programming languages provide a continuation character that is used to indicate that the statement continues on the next line. Any factors that increase the readability of the source code, and are part of the code, are classified as intrinsic documentation.

GROUP TASK Research

Search the Internet for code samples in a variety of different high-level languages. Comment on the intrinsic documentation used. What identifier naming conventions have been used?

Other forms of technical documentation

Throughout the software development cycle, various forms of documentation are created. These models, diagrams and documents should be revised and edited to reflect any changes made as the software development cycle has progressed. For example, the design specifications that were originally created in Stage 1 will no doubt have undergone modification and refinement as the project's development has unfolded. Any modifications that have been made should be documented and retained with the original specifications. CASE tools used during the software development process, assist in ensuring technical documentation is updated to reflect the final nature of the product.

System models, data dictionaries and algorithms must be particularly accurate. Future maintenance developers will require these documents.

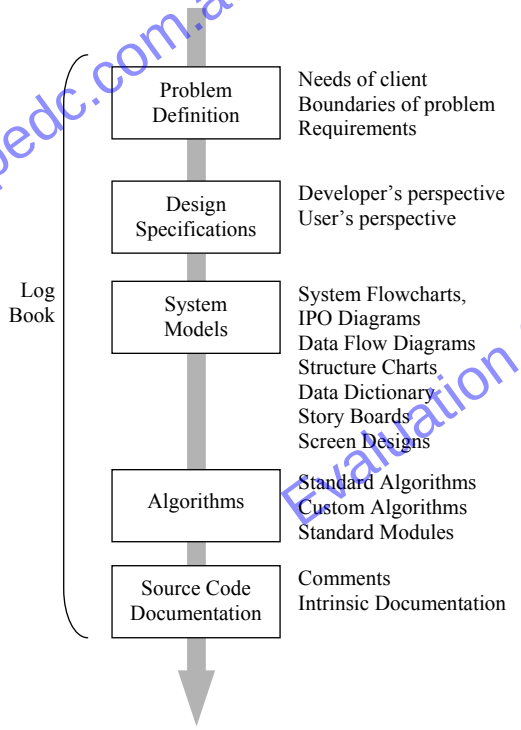


Fig 5.43
Technical documentation is compiled throughout the software development cycle.

The technical documentation we have examined in the SDD HSC course and that is created during the development of many software products is described in *Fig 5.43*.

GROUP TASK Activity

Make up a table listing all the possible technical documentation types studied so far in the text. Briefly describe each using diagrams as examples where appropriate.

HSC style questions:

1. Which of the following forms of documentation would be most useful to a maintenance programmer?
(A) Online help
(B) Gantt chart
(C) Story board
(D) Structure chart

2. Which of the following are all examples of user documentation?
(A) Installation guide, reference manual, system models and design specifications.
(B) Reference manual, tutorial, online help and intrinsic documentation.
(C) User manuals, tutorials, online help and installation guides.
(D) Online help, tutorials, system models and user manuals.

3. Which of the following best describes the purpose of internal documentation?
(A) To improve readability of source code.
(B) To improve the efficiency of code execution.
(C) To provide assistance to users of the software.
(D) To protect the intellectual property rights of the developer.

4. A software developer uses software that simulates many users entering data into their application simultaneously. This software is an example of which of the following?
(A) COTS package
(B) CASE tool
(C) Reverse engineering
(D) Test data

Question 5.

You have been contracted to make modifications to an existing software application that you did not write. Outline TWO examples of documentation that would be useful when making the modifications. Justify your choices.

Suggested Solutions

1. (D) **2.** (C) **3.** (A) **4.** (B)

Question 5.

Internal documentation incorporated into the source code which includes comments and also intrinsic documentation such as meaningful variable and procedure names. Internal documentation will make it easier for the programmer to understand exactly what the source code is doing, which makes it easier to determine which parts of the code need modifying to achieve the specified changes.

Systems documentation such as systems flowcharts, structure charts and DFDs provide an overview of the system and how the various parts interrelate. This provides the programmer with the information to determine the role played by individual components of the system, so they can better determine which parts need to be modified, and the effects of those modifications on other parts of the system.

SET 5E

1. Which type of documentation includes hardware and software requirements together with step-by-step set-up instructions?
   (A) User manual.
   (B) Reference manual.
   (C) Installation guide.
   (D) Tutorial.
2. Internal documentation is:
   (A) included in the source code.
   (B) comments used to explain the source code.
   (C) self documenting code.
   (D) All of the above.
3. The complexity of the language used in documentation should be determined by:
   (A) identifying the intended audience.
   (B) the nature of the product.
   (C) the complexity of the code.
   (D) examining similar products.
4. Examples of user documentation include:
   (A) installation guides, process diaries, reference manuals and user manuals.
   (B) installation guides, internal documentation, tutorials and user manuals.
   (C) user manuals, reference manuals, tutorials and system models.
   (D) user manuals, reference manuals, tutorials and installation guides.
5. Intrinsic documentation includes all of the following:
   (A) Meaningful identifier names, indenting within control structures, use of colour and spacing.
   (B) Comments, meaningful identifier names and indenting within control structures.
   (C) Internal documentation, algorithms and system models.
   (D) Objectives, data dictionaries, system models and feasibility studies.
6. A document listing all the commands available in an application is known as a(n):
   (A) User manual.
   (B) Reference manual.
   (C) Installation guide.
   (D) Tutorial.
7. A chronological record of the tasks undertaken during the development of a software solution could be best described as:
   (A) internal documentation.
   (B) a log book.
   (C) intrinsic documentation.
   (D) a system model.
8. A form of documentation that uses sample data to instruct the user in the use of a software product.
   (A) User manual.
   (B) Reference manual.
   (C) Installation guide.
   (D) Tutorial.
9. The sentence "The aim of this game is to capture all the opponents Greebs" is an example of a:
   (A) procedural explanation.
   (B) subjective explanation.
   (C) technical explanation.
   (D) conceptual explanation.
10. Comments within the source code are used to explain the code. Which of the following is true in regard to these comments?
    (A) They should explain the logic of the code that follows.
    (B) Each line of code should be preceded by a comment.
    (C) They should explain what a section of code does rather than how it does it.
    (D) Each comment should precisely reflect the processing of the code fragment that follows.
11. Make a list of all the user documentation types. Describe the main features of each.
12. Make a list of all the different forms of technical documentation. Describe the main features of each.
13. 'Documentation should be developed once the product is ready for distribution.' Do you agree with this statement? Explain your answer.
14. Electronic forms of user documentation are becoming commonplace. Explain the advantages of electronic forms of user documentation over traditional printed forms. Consider the point of view of the software developer and the user as part of your response.
15. Create a subroutine in a high-level language with which you are familiar, that splits a sentence into separate words. The words are to be stored in an array. Include suitable internal documentation as part of your solution.

HARDWARE ENVIRONMENT TO ENABLE IMPLEMENTATION OF THE SOFTWARE SOLUTION

All software solutions are designed with particular hardware requirements in mind. The installation guide will include specifications in regard to minimum requirements, possible additional hardware and appropriate drivers. Large custom systems are usually designed for a specific hardware configuration. Solutions designed for a broad market need to operate on a variety of hardware configurations. The hardware requirements need to be specified and tested before software solutions are released.

Minimum Hardware Configuration

The minimum hardware requirements will vary for different software solutions. The checklist, shown in *Fig 5.44,* lists hardware items that need to be considered by the software developer for all solutions. The minimum configuration should allow the software to operate successfully with acceptable performance and response times. *Chapter 6, Testing and evaluating software solutions*, examines how the hardware and software is tested to ensure performance meets expectations.

- Processor type and speed
- RAM size and type
- Secondary storage size and access speed
- Input devices
- Output devices
- Network hardware

Fig 5.44
Checklist of hardware items required for all software solutions.

Possible Additional Hardware

Additional hardware items are those items supported by the software product but not essential for its operation. Extra RAM may allow the product to operate with larger files. Sound cards may allow for optional audio feedback for the sight impaired. The addition of a network or internet connection may allow for foreign exchange rates to be updated electronically in financial packages.

Appropriate Drivers or Extensions

Hardware devices require software drivers to allow them to communicate with the system. These drivers are also known as interfaces, as they convert signals from one device into those that can be understood by another. Before a hardware device can be used by a software solution, its driver must be correctly installed and configured. Most peripheral devices come packaged with drivers for most popular operating systems or are able to use drivers included with the operating system. The driver is installed as part of the installation of the hardware device.

Custom built hardware may require a purpose built driver to be created. This driver would need to be installed as part of the installation of the software solution. In this case, the driver is an extension to the software package. For example, a product designed to monitor the climate within a green house includes custom drivers for the temperature and humidity sensors. These sensors are part of the minimum hardware requirements for this product. The drivers or extensions are installed as the main application is installed.

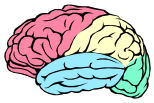
GROUP TASK Investigation

Examine the hardware specifications for a number of commercial software applications. Classify each as either a minimum requirement or an additional hardware item.

EMERGING TECHNOLOGIES

New advances in both hardware and software continue to have a profound effect on our daily lives. The implications of these technologies will have far reaching effects for software developers. Software developers must remain up to date with current and emerging technologies if they are to remain competitive. The Information Technology industry is renowned for its speed of evolution.

Let us consider some emerging hardware and software technologies that have the potential to alter both the human environment and the software development process.



Quantum Computers

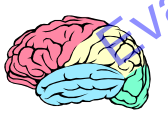
Quantum theory postulates that matter can be in more than one state at the same time. For example, a mouse held in a sealed box could be either dead or alive. We don't know the state of the mouse until the box is opened. Quantum theory suggests that whilst in the box, the mouse is effectively both dead and alive. Strange as this may seem, science has found that photons, a component of light, actually behave in this manner. Photons behave as waves and at the same time behave as particles. Quantum physics defies the laws of classical physics, however it does explain the consequences of nuclear reactions and has helped unravel the mysteries of DNA.

The theory of quantum computing relies on spinning particles. Many particles at the atomic level are known to spin. The direction of the spin can be used to represent binary digits (bits). In quantum computing terms, bits are known as qubits (pronounced cubits). We are not only able to set the direction of the spin but we are also able to read the direction of the spin for an individual qubit. Using the 'mouse in the box' analogy, if we place particles in a box out of eyesight and apply energy, each particle will be in two states at the same time. If we have 8 particles in the box, we are representing 2^8 or 256 different states at the same time. Applying an algorithm to the particles allows calculations to therefore take place on all 256 states concurrently. If we expand our theory to many thousands or millions of particles, the potential processing ability becomes mind-boggling.

Although quantum computers are still very much a theoretical possibility, there are many companies investing heavily in their research and development. Why the investment? One very practical use revolves around security of data. Current methods of encrypting data rely on the fact that breaking the encryption is not possible with today's processing capabilities. It is said that if all the computers in the world worked continuously on breaking just one modern military encryption code it would take about the same amount of time as the universe has been in existence. Quantum computers could potentially break encryption codes in an instant.

GROUP TASK Discussion

Quantum computers, if they become a reality, will revolutionise the computing industry. How do you think the availability of virtually unlimited processing power will affect the software development industry?



Human-computer interaction

Research continues to improve the interaction between humans and computers, the aim being to simulate human communication using computers. Humans communicate using eye movements, facial expressions, touch and voice. These inputs combine to create each human message. The challenge for researchers is to create systems that are able to mimic this process.

Projects specialising in artificial vision are currently able to follow the eye movements of a user. Eye movements are analysed and used to move a cursor on the computer screen or alter the view appropriately as the user moves through a virtual world.

Users can direct computers and game consoles using body movements. Video cameras and other sensors are used to detect even small hand and finger movements. The software responds intuitively. For instance current dance games are able to detect timing and small body movements and shooting games can accurately determine the precise direction a person is pointing.

Output using gloves and full body suits incorporating pneumatic sensors is becoming a reality using lighter and more responsive systems. The computer opens and closes a series of valves within the glove or suit such that the user can feel objects as they are manipulated in the virtual world.

Voice recognition is continually being refined. For instance early mobile phone voice recognition was rudimentary and often failed to detect single words correctly. Today complete sentences are understood with amazing accuracy. Devices are under development that detect lip movements, causing directional microphones to home in on the vocal source. These devices are able to operate in high noise environments or where there are groups of users.

Fig 5.45
C-Pen text scanning pen.

The following devices all introduced new methods for human-computer interaction:

- Games console devices such as the Wii remote, PS3 move, Xbox Kinect which include various motion, video, sound and other sensors within an integrated system.
- Handheld communication devices such as the iPhone and other smart phones.
- Biometric devices such as finger print, iris, face recognition and other biometric scanning devices.
- Touch sensing devices including touch screens, multi-touch surfaces, movement sensors and touch pads.
- Scanning devices such as scanning pens for text, phone cameras used to scan QR (quick response) codes, smart card readers within EFTPOS terminals or RFID scanners used for stock control in warehouses, retail stores and libraries.
- Mind control devices such as Emotiv's Epoc Headset which is able to detect facial expressions, emotional state and cognitive processes using 14 sensors.

GROUP TASK Discussion

Examine the games consoles, smart phones and other devices owned by members of the class to identify all the input and output devices present.

GROUP TASK Activity

Identify all the various scanners you encounter during a typical week. Briefly explain how each scanner improves the ability of software to collect data compared to keyboard data entry.

GROUP TASK Discussion

Emerging methods of input and output will no doubt continue to have a profound effect on future software products. Discuss the effects you perceive these changes may have for the development of future software solutions.

Internet access initiatives

New devices and software applications are regularly released which allow more flexible access to the internet and which utilise the internet in many new and innovative ways.

Examples of newer internet related hardware and software initiatives at the time of printing include:

- Televisions, mobile phones, cameras, cars, white goods and other devices which increasingly include built in World Wide Web connectivity.
- Social networking software which operates on a wide range of devices and is integrated within a wide range of other software applications.
- Location based applications that utilise GPS coordinates and internet connectivity to map locations in real time.
- Wireless connectivity at ever increasing speeds which allows people to remain connected regardless of their location or activity.
- The National Broadband Network (NBN) which is planned to provide optical fibre to the premises of most homes and businesses throughout Australia.

GROUP TASK Discussion

Investigate new emerging internet technologies. How do these technologies affect human communication? What consequences are there for software developers working on internet solutions?

GROUP TASK Research

Use the internet to research new and innovative computer based devices. Write a paragraph on at least two such devices. Compile a class summary of all the devices found.

CHAPTER 5 REVIEW

1. Consistent user interfaces allow users to:
   (A) transfer skills to new products.
   (B) concentrate on the tasks at hand.
   (C) predict the results of their actions.
   (D) All of the above.

2. EBNF and railroad diagrams:
   (A) are used to describe algorithms.
   (B) describe the syntax of programming languages.
   (C) are important forms of user documentation.
   (D) can be altered by the programmer to suit the particular project.

3. Parsing is a process used during syntactical analysis. Parsing ensures:
   (A) each statement contains elements that are legal parts of the language.
   (B) source code is free of logic errors.
   (C) that source code cannot be changed by the end users.
   (D) each statement obeys the rules of the high-level language.

4. In regard to the role of the CPU in the execution of software, which of the following is true?
   (A) Assembler code is the native language of the CPU. Each assembler instruction corresponds to a precise hard-wired process.
   (B) Each high-level language calculation corresponds to one implementation of the fetch-execute cycle.
   (C) Microcode instructions are used to activate one implementation of the fetch-execute cycle.
   (D) The control unit translates high-level instructions into microcode.

5. The source code must be distributed with a particular product. What is the most likely reason for this?
   (A) The program must be translated using a compiler.
   (B) The software needs to be modified by the user.
   (C) The program is written for translation using an interpreter.
   (D) The source code is in the public domain.

6. Source code need not be distributed with the final product. Which of the following could be used?
   (A) Interpretation.
   (B) Compilation.
   (C) Byte code or other intermediate code.
   (D) Either B and C.

7. A program requires the user to make a selection from a total of 20 items. The most appropriate screen element to use would be a:
   (A) series of check boxes.
   (B) series of radio buttons.
   (C) list box.
   (D) menu.

8. If the RAND function returns a decimal between 0 and less than 1 then RAND*21+4 will return a value from:
   (A) 0 to less than 25.
   (B) 4 to less than 21.
   (C) 4 to less than 25.
   (D) 0 to less than 4.

9. Dynamic link libraries (DLLs) are used to add functionality to high-level languages. Which of the following is true of DLLs?
   (A) DLLs must be distributed with the final software product.
   (B) DLLs are combined within the executable file by the linker.
   (C) DLLs are only required for interpreted programs.
   (D) Each DLL must be present on the user's machine for the software product to execute correctly.

10. Hardware requirements for new software products need to be specified. Why is this the case?
    (A) Software products are often designed for specific combinations of hardware configuration.
    (B) It is often not possible for software developers to predict the hardware environment on which their products will be installed.
    (C) Software solutions will operate more efficiently under particular hardware conditions.
    (D) All of the above.

11. Translation of source code into executable files is necessary for all software solutions. Explain the processes involved in this translation process.

12. The fetch-execute cycle takes place for every task completed by a computer. Explain the processing that takes place to execute a simple high-level language assignment statement such as Count:=1. Answer in terms of the operations that take place within the CPU including references to the fetch-execute cycle.

13. Program development techniques aim to produce efficient and error free software solutions. Describe at least five techniques that are available to software developers that assist them in this task.

14. Select a new input device such as a recent game controller, scanning device, biometric device or touch device. Research how the device operates and how to collect data from the device within source code.

15. New technologies are constantly appearing. Using the internet, explore the most recent advances in CPU technology. Compare the speed and processing capabilities of these new processors with those available ten years ago. Do you think processing speeds and capabilities will continue to evolve at such a large pace?